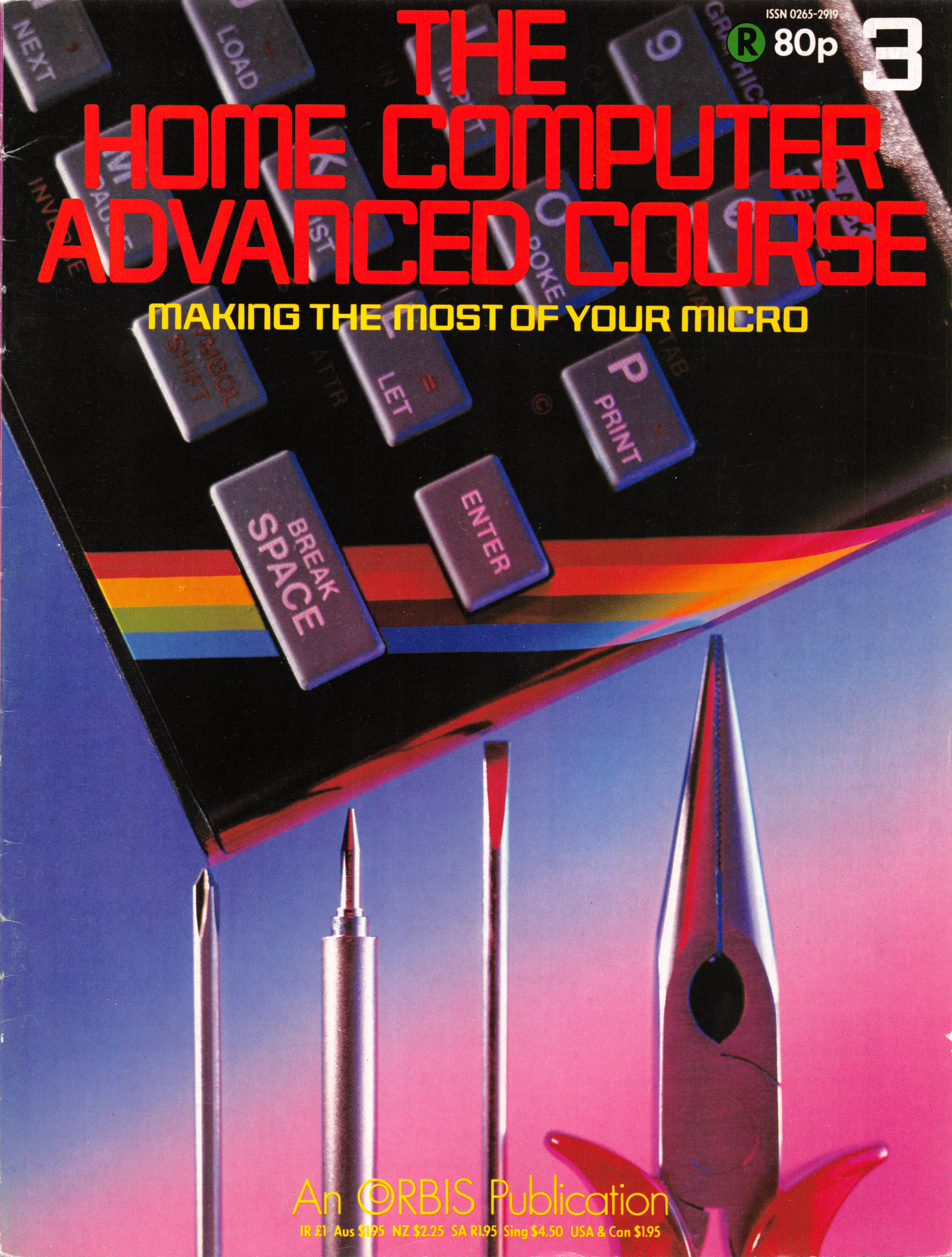
ISSN 0265-2919

80p

3

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO

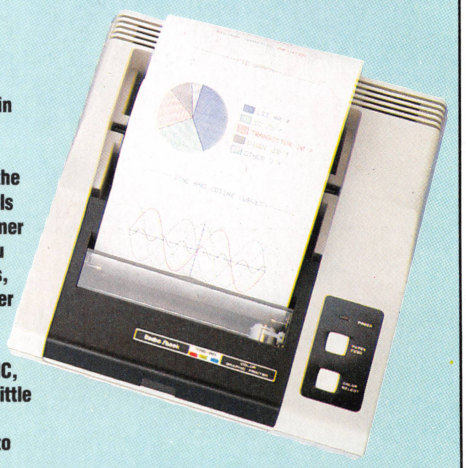


An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week



- The second part of our review of educational software currently in use in our schools.
- Printers can be one of the most expensive peripherals the average computer owner is likely to buy. Unless you want letter-quality results, however, there are cheaper systems available.
- Like all dialects of BASIC, the BBC's version has its little quirks and command structures that you need to come to terms with.

Editor Jonathan Hilton; **Art Director** David Whelan; **Deputy Editor** Roger Ford; **Production Editor** Catherine Cardwell; **Staff Writer** Brian Morris; **Picture Editor** Claudia Zeff; **Designer** Hazel Bennington; **Sub Editors** Robert Pickering, Keith Parish; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Researcher** Helena Siedlecka; **Contributors** Lisa Kelly, Steven Colwill, Richard King, Martin Hayman, Roger Munford; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1;

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.

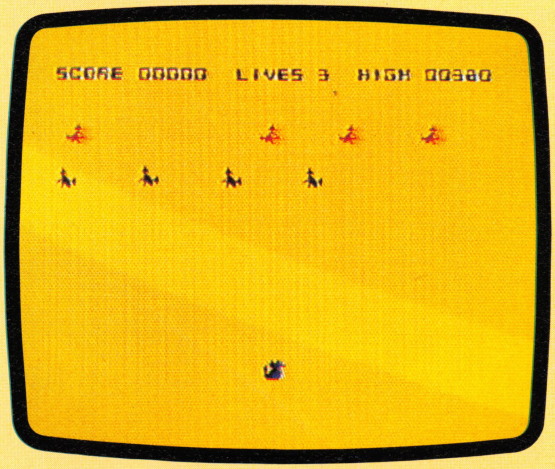
Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Details of how to obtain your binders (and of our special offer) are in issue 5. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

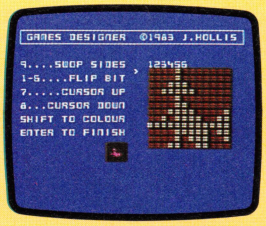
ACTION REPLAY

The Games Designer

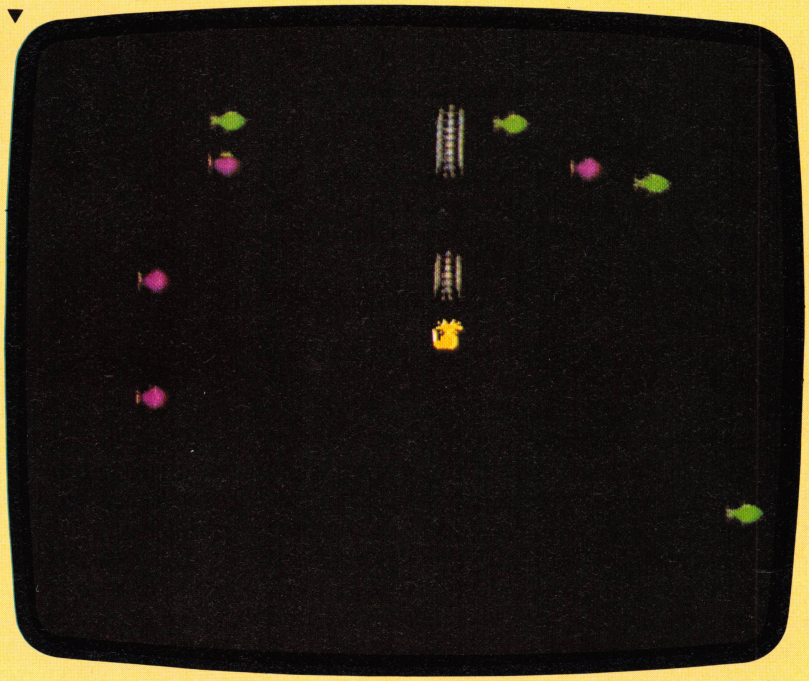


Standard Format
The sprites will initially appear against a plain background, following a slow and orderly descent

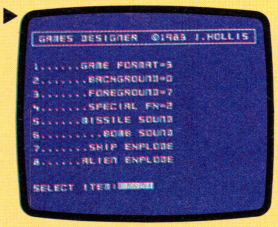
Final Effect
This shot shows the changes to the original sprites and background colour. The only element unchanged is the cat



Sprite Configuration
Select the appropriate co-ordinate on the chart you want to fill (or erase). At the bottom of the screen you can see the actual shape and colour of the sprite



Configuration Menu
This menu gives you the option of selecting the direction of movement of your ship or laser base, the foreground and background colours, how the aliens appear, and sound effects for all elements



Movement Menu
This allows you to determine the attack approach. At the top right of the screen you will see a numbered direction chart and at the bottom right there is a pattern display where you can monitor the effects

IAN McKINNELL

Millions of microcomputer games packages are sold every year and the size of the market for this ready-made software was an indication to enterprising software houses that there was good sales potential for games generator packages. What is surprising is that so many of the games on the market are similar to one another.

Most games played on home computers fall into two categories: Adventure participation games, with or without a graphic representation of the scenario on the screen, as in The Hobbit; and the totally screen-based 'fantasy simulators' such as Space Invaders or Asteroids.

Even a cursory analysis of the two generic types reveals the reason for their similarity. To take the fantasy simulators first, there are two prerequisites of the Space Invader type of game — a firing base and a target to aim at. So if we can construct abstract versions of these two and allow the games designer to decide independently on the defender's position and fire-power and the frequency and intensity of the attack waves, then it is feasible to produce a variety of games, each subtly different from the next, simply by varying the parameters. An analysis of the numerous games with similar-sounding names produced within a games 'generation' will reveal that this philosophy has been applied by the professional software producers.

A prime example of a software package that allows you to do just this is John Hollis's Games Designer (from Software Studios/Quicksilva for the Spectrum 48K). Games Designer offers you eight outline games. Having selected one of them you can change all the basic parameters mentioned above, but it does not allow you to build up a new game from scratch.

The package is menu-driven, and even as a series of pre-programmed games, the Games

Designer is good value for money.

Having chosen a game to be the outline for modification, you are presented with a choice of the type and scope of that modification, starting with the shape and colour of the sprites used to represent the protagonists. Even though a menu of existing sprites is presented, this is more a convenience than a constraint, for you are able to start from a predefined design and change any bit within the 12 × 12 matrix, or opt to begin with a 'blank page' and originate a completely new figure. The 12 × 12 format (two characters by two characters) results in a perfectly usable and distinctive sprite.

Sprite definition itself is menu-driven, the choice of available commands being displayed to the left of the screen and a magnified image of the sprite in question at the right. A helpful addition is a second image of the sprite, displayed in its actual size and colour at the foot of the text.

The next step is to define the manner in which these sprites will move about the games ground. This section, called Configuration, actually allows you greater scope than simply directing the movement of individual tokens. Following on from this, in the order in which options are displayed on the main menu, you are invited to select a movement pattern for the screen tokens. To make the movement more complex, two or more patterns may be linked together at this stage.

Now you have to decide on the frequency of the attack waves and insert special effects, both visual and audible.

Having created a game, it only remains to save it on cassette, so that it can be played again in the future. Here the package falls down badly, from the user's point of view, for it does not store a game in playable form, but only as input parameters to the Games Designer package itself, rendering the exercise useless to those who might wish to develop a game for subsequent sale.

THE QUILL

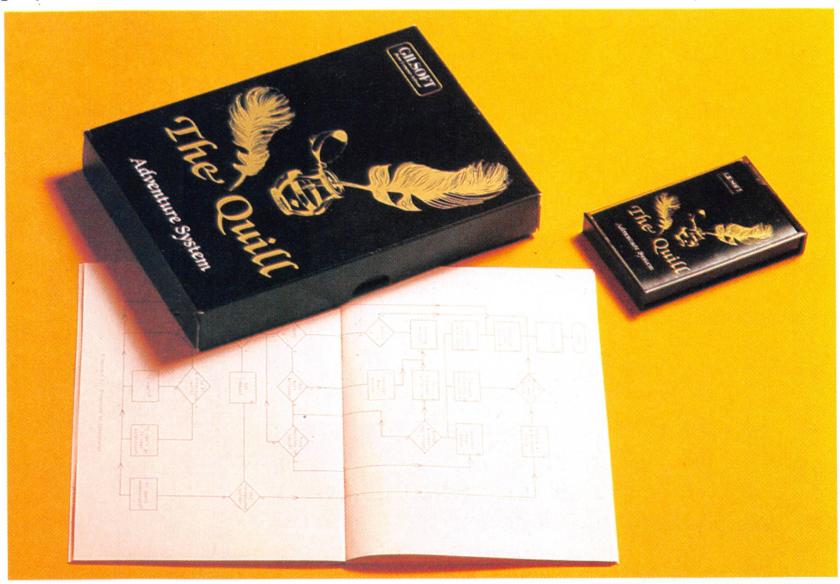
In direct contrast to Games Designer, The Quill is oriented towards those who wish to create Adventure games and attempt to market them — there is even a section in the comprehensive manual headed *Selling your Adventure* that gives useful hints on the steps you should take in testing and debugging. Chief among the hints is to have as many people as possible play the game critically. Remarkably, the authors require nothing more than a 'mention somewhere in [the game] that it was written with The Quill' — a rare piece of altruism.

While it is presented as a generator of Dungeons and Dragons type games, the Quill uses techniques more akin to those employed by commercial database management packages. It is divided into three main parts — the database itself; a database editor, which allows the parameters to be set up; and a database interpreter, which executes the game interactively.

Just as one may analyse Space Invaders and the like as protagonist-positioning games, so the basis of Adventure games is the 'conceptual maze': a maze that exists in more dimensions than the purely spatial ones. Thus, in addition to seeking the exit route the player is presented with a list of objects, each of which is useful only in a specific circumstance — if you never go into a room with an empty light socket, then there is no point in carrying a light bulb, for instance.

To make the game more interesting, a limit is normally placed on the number of objects the player can carry with him from place to place — and there is nothing to stop the designer using this artificial limit to send the player back and forth accumulating the collection of objects that he will need to accomplish the tasks he is set as the game unfolds.

Defining The Object
The Quill offers users a way of creating Adventure games for themselves. Similar in style to a number of database management packages, the Quill requires the locations and objects that feature in the adventure to be defined. Object oriented programming techniques then allow these assigned attributes to be called up and displayed, or used as parameters, whenever the object or location features in the game



IAN McKINNELL

The first stage in constructing an Adventure game is to map out a script and a scenario in which the script will operate. The Quill's manual starts by outlining a simple game which presents the player with a choice of ten objects in a simple environment comprising six rooms. This game is not embodied within the program, forcing you to enter all the parameters outlined in the manual before you can play it, just as you would have to if you created your own game.

The Quill places a limit on the number of parameters that may be used in the game, but this is so large — 252 locations and 210 objects — that you are unlikely to run out of options.

The package is quite capable of producing Adventure games to rival most of those commercially available (in fact a number of them were written on it), but it does have its limitations. First, it does not allow the creation of screen graphics such as those used to good effect in The Hobbit, for example; nor does it permit any interaction with characters — so there is no point in trying to create them. Apart from offering the Adventure games player a simplified method of composing his own games, its great strength lies in the discipline required to use it.

STATE OF THE ART

Since their appearance in 1982, Motorola's 68000 series of microprocessors, the large-capacity successors to the much acclaimed 6809 eight-bit CPU, have captured a significant part of the market for 16-and 32-bit processors. Selected by Sinclair to power the QL, the 68000 is certain to dominate 16-bit computing.

The 68000 bears a strong resemblance to the MC6800, an earlier Motorola microprocessor that is still widely used, particularly in peripherals such as intelligent controllers. This means that the much more capable 68000 is simple to interface and has the support of a wide variety of ready-made hardware. This includes I/O boards with 6821 PIA (Peripheral Interface Adaptor) chips, VDUs with 6845 CRTCs (Cathode Ray Tube Controllers), clocks using the 6840 programmable timers, and disk controllers.

Another feature of the 68000 makes it attractive to computer designers, and that is the width of the data and address buses. These are completely separate from each other, and each bit has its own pin — unlike the 8086, 8088 and Z8000, on which the pins are multiplexed together: the two buses share a set of pins, signals being interleaved and decoded at their destination.

The processor can therefore run as fast as the rest of the system will permit, and with the newer 50- or 90-nanosecond (10^{-9} seconds) RAM chips this can mean a reduction, or even an elimination, of wait states. The fastest processor in the 68000 series is the 68000L12, many aspects of which can be run at 14 Megahertz.

Sinclair Research has used the 68000's successor , the 68008, in the QL. Internally this is much the same as the others in the series, but to make it more compatible with existing eight-bit systems it has an eight-bit data bus rather than the full-width 16-bit bus. Since it needs fewer pins it comes in an ordinary 40-pin package.

Motorola is soon to produce an even more powerful microprocessor. The 68020 is a 32-bit microprocessor that needs a 96-pin package, the shape and style of which have yet to be finalised. The 68881, a specialised floating point maths processor with eight registers (each 80 bits wide) that will greatly increase the amount of 'real' data that can be handled, is also being planned.

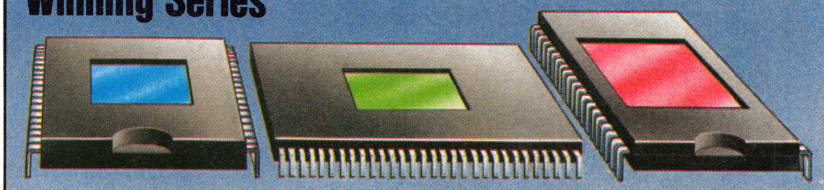
A number of other chips in the 68000 series provide I/O functions similar to those found in earlier chips, but greatly enhanced. From the programmer's point of view, however, the 68000 has many advantages over most other 16-bit processors, owing to the symmetry of its address and data registers and the rich instruction set.

It isn't perfect, though. First, a distinction is made between the address and data registers, though they are the same size (32 bits) and in most respects are operated upon in the same way by the same instructions. As a result, it is often necessary to move data from an address register to a data register, manipulate it, and then return it to the address register. It would have been easier if Motorola had allowed any register to be used for either data or addresses.

Second, there are redundancies in the instruction set, but as these result from what might be called 'addressing mode cross-over', this isn't critical. This phenomenon arises because the various addressing modes are so different that sometimes one means exactly the same as another, despite having been arrived at by different instructions.

In general, however, the Motorola 68000 series provides large, fast and efficient CPUs that are becoming widely used. In the past year they have been used in Apple's Lisa and Mackintosh, Sinclair's QL, and many multi-user business machines with lower profiles in the market. Providing features that would have cost thousands of pounds only a couple of years ago, and available at a reasonable price, they seem set to become as popular among the next generation of machines as the Z80 and 6502 are today.

Winning Series



6502
Developed by MOS Technology, the 6502 microprocessor was to become, with Zilog's Z80, the mainstay of the microcomputer industry. It utilises a 16-bit address bus and an eight-bit data bus. Chief amongst its peculiarities is the organisation of its registers. There is only one accumulator, but the whole of memory page 0 can be used as general-purpose registers

68000
Motorola redesigned and redeveloped the 6800 into the 6809, but not in time to secure a large share of the eight-bit market for itself. This was to prove an advantage, for it led the company to develop the 68000 16/32-bit processor. The 68000 is able to utilise many of the 6502/6800 series support chips, and is built around eight 32-bit data registers and seven 32-bit address registers

Z80
Theoretically more powerful than the MOS 6502, the Zilog Z80 uses similar address and data bus structures but has a considerably strengthened register set — 12 eight-bit general-purpose registers and two 16-bit index registers — and a much larger instruction set. Perhaps its greatest advantage over the 6502 was its ability to support the CP/M Operating System

KEVIN JONES

CRASH COURSE

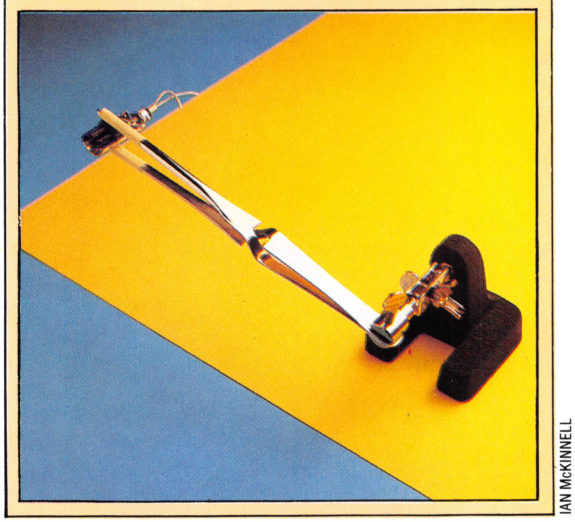
Microcomputers and hi-fi audio have many things in common. To the user there is a further — and often annoying — point of similarity: the high cost of installing small ancillaries and accessories. This series starts with soldering, to enable the complete beginner to do for himself those little jobs that otherwise might cost pounds.

Soldering, like brazing, is a method of joining two metal objects together by means of a soft (hence, easily melted) metallic alloy; lead and tin are used for electronics work. The objects to be joined are themselves heated to a temperature higher than the melting point of solder (280°C/535°F), solder is applied to the components, they are brought into contact, and the heat source is removed. As they cool down, the solder solidifies and the joint is made.

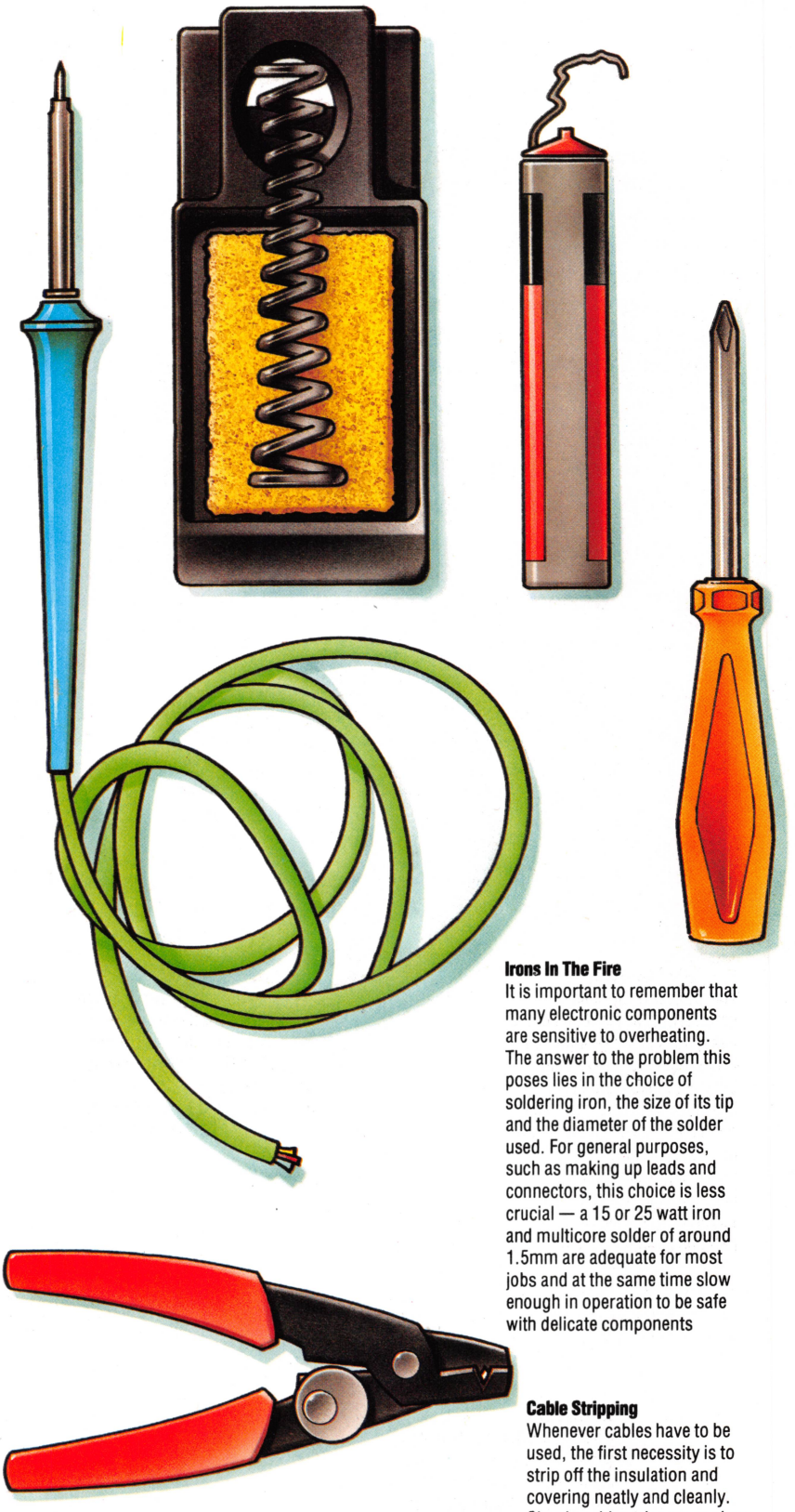
A soldering iron applied directly to a stick of solder will melt it almost immediately. The hot solder will cool down almost immediately on coming into contact with the cold component. The result is known as a 'dry' joint. At best, it will not hold together. At worst it will make a very poor connection, perhaps even allowing an intermittent flow of electricity. There is only one way to prevent this imperfect joining: heat the component until the solder melts on contact.

Useful Vices

One of the reasons why computers are part of our everyday lives is their tiny size. Their component parts, then, are so small that working with them can be awkward. There are a variety of lightweight vices and grippers available at reasonable cost. If you find the grip insufficient, try making up sleeves of masking tape, sticky side out, to slip over the forceps-like jaws

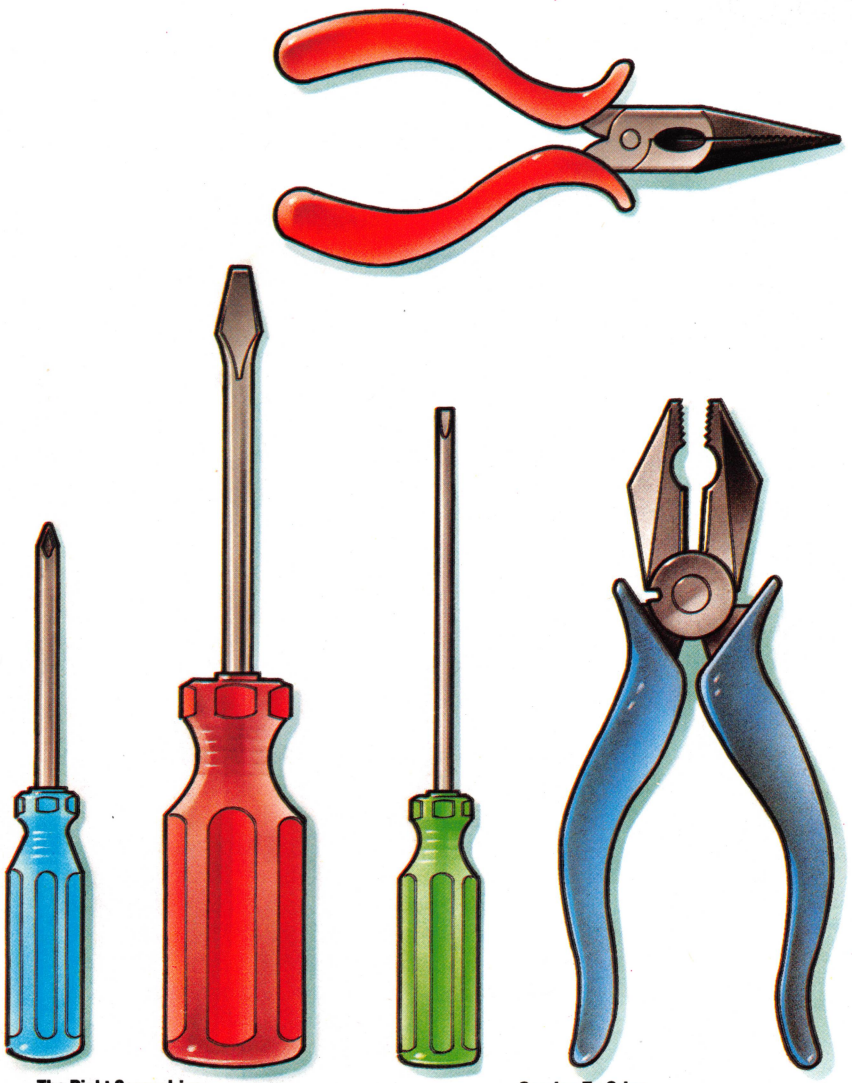


IAN McKINNELL



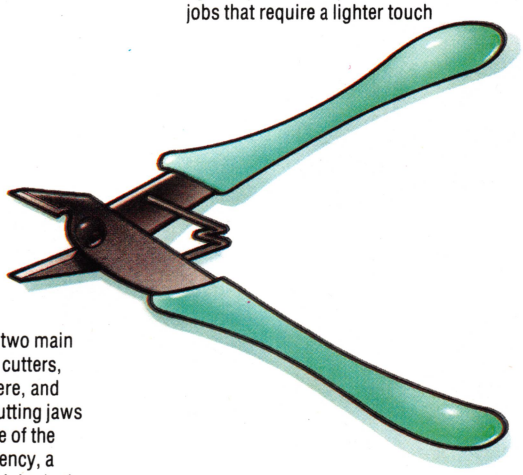
Irons In The Fire
It is important to remember that many electronic components are sensitive to overheating. The answer to the problem this poses lies in the choice of soldering iron, the size of its tip and the diameter of the solder used. For general purposes, such as making up leads and connectors, this choice is less crucial — a 15 or 25 watt iron and multicore solder of around 1.5mm are adequate for most jobs and at the same time slow enough in operation to be safe with delicate components

Cable Stripping
Whenever cables have to be used, the first necessity is to strip off the insulation and covering neatly and cleanly. Simple cable strippers such as the one shown here cost very little, and can be preset to a given cut depth, removing the insulation cleanly but leaving the cable itself intact



The Right Screwdriver
There are two varieties of screwdriver: straight and cross-head, though in the latter case it is necessary to differentiate between Philips and Pozidriv. However, in the size range used on most home computers, the two standards are interchangeable. You will probably find you need both straight and cross-head, in perhaps a range of sizes

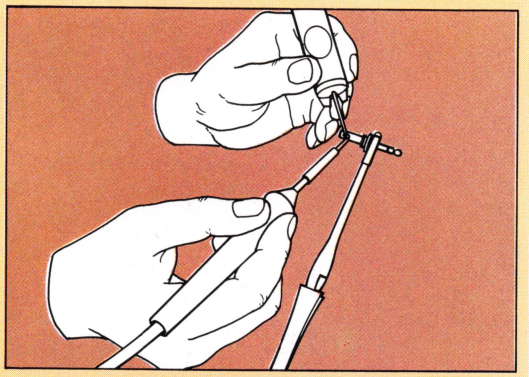
Coming To Grips
For the beginner two types of pliers are sufficient — heavy duty bull-nose pliers, which will probably double as wire cutters for the heavier jobs, and much lighter, point-nose pliers for jobs that require a lighter touch



Cut-off Point
Wire cutters come in two main types, known as side cutters, like the pair shown here, and end cutters, whose cutting jaws are at 90° to the plane of the handles. In an emergency, a pair of nail clippers might do the job — but don't rely on being able to cut your nails with them afterwards!

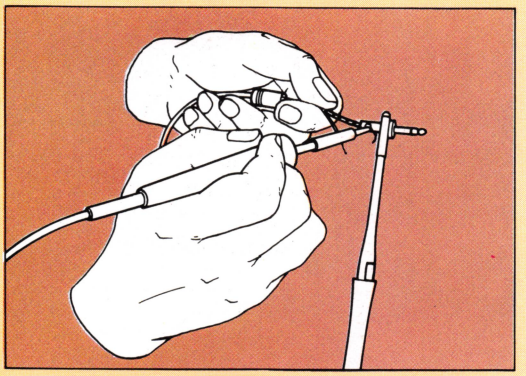
MICHAEL BROWNLOW

Soldering A Plug

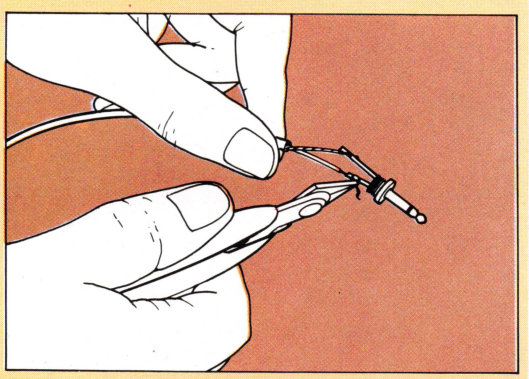


Baring And Tinning
The first stage in the process of making up a lead is to bare the wires in the cable by stripping off the insulation. Be generous. Strip more wire than you need, and trim it back later. Hold the cable in the vice and heat it with the iron. Apply the solder to the cable, and when it starts to run use the iron to lead it neatly down the whole exposed length of the cable. This process, known as tinning, makes the later soldering process considerably simpler — by then, the solder is on the component.

Repeat this same procedure on the plug. While the terminal is still hot enough to run the solder, apply the tinned cable, remove the heat source when the solder on both component and cable have fused and run together, blow on them to cool them below melting point, and the joint is made



Keeping In Trim
The last stage of the job is to trim off waste material. Trim cables short, and also the terminal pins on components — but not until the very last moment, and that means after the job has been tested. The reason for trimming is simple — a long end of waste cable, or a protruding terminal pin, can brush up against something else, causing a short circuit or, worst of all, one of those infuriating intermittent faults

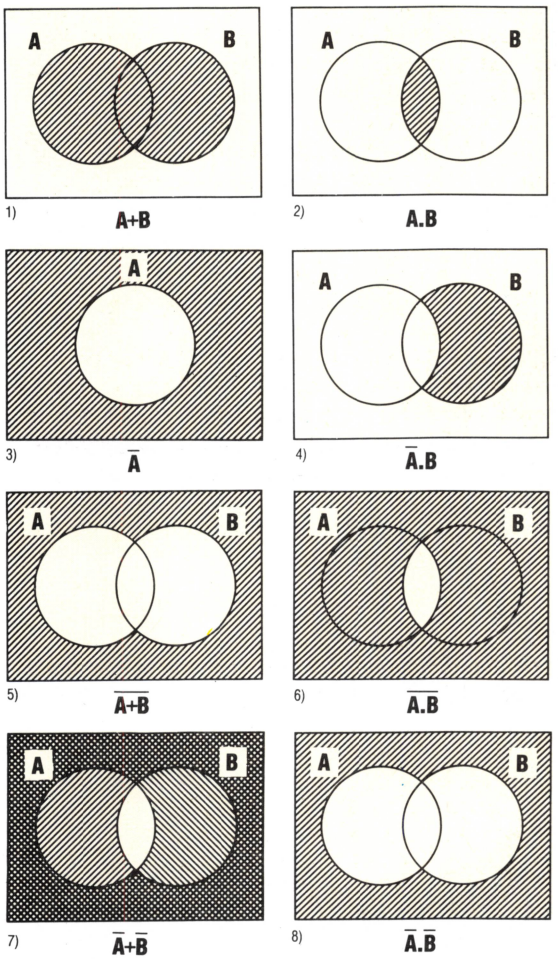


KEVIN JONES

REFINING THE PROCESS

Simplification of Boolean algebra expressions involves the production of equivalent expressions that contain fewer operators (AND, OR or NOT). This simplification is of major importance in logic circuit design as it provides clearly defined methods by which the circuit may be improved in terms of layout and economy.

Venn diagrams provide a useful graphical aid to the simplification of Boolean algebra expressions by allowing simple expressions to be drawn as areas of shading. The area inside a rectangle (symbolised by 1 = Identity or Universal Set) represents all the possible combinations of truth values of the inputs, and circles within the rectangle represent certain combinations. Here we show some represented as Venn diagrams:



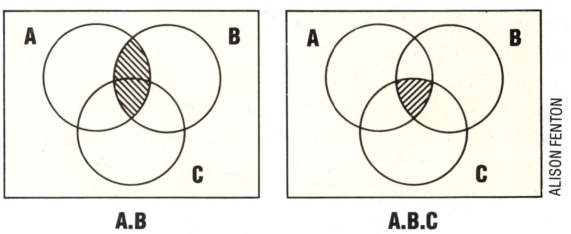
Comparing diagrams 5 and 7 shows at a glance that NOT(A OR B) is not the same as NOT(A) OR NOT(B). Similarly, diagrams 6 and 8 demonstrate that NOT(A AND B) is not equivalent to NOT(A) AND NOT(B).

Perhaps the easiest way to think of AND and OR in terms of Venn diagrams is to think of A.B as the area where area A 'overlaps' with area B; and to regard A+B as the combined areas of A and B. There are several self-evident relations that exist in Boolean algebra. For each of these relations you may try to construct a Venn diagram as proof that they are true. (0 represents the 'empty set' — that is to say an impossible combination.)

1) $A.A = A$
2) $A.\overline{A} = 0$
3) $A.0 = 0$
4) $A.1 = A$
5) $A.(A + B) = A$
6) $A.(\overline{A} + B) = A.B$

LAWS OF BOOLEAN ALGEBRA

The concept of *duality* is an intriguing and useful aid to simplification, relying on the symmetry of the operators AND and OR. To form the dual of any true relation change all the ANDs to ORs and vice versa, and likewise all the noughts to ones and vice versa. For example, let us take the fifth relation mentioned in the preceding list. The dual of this relation is $A + A.B = A$. This expression is also true. It demonstrates another important principle, namely *absorption*. Looking at a Venn diagram, it is clear to see that the A.B term lies wholly within A, and thus can be said to have been absorbed by A. This idea can be extended to a three-variable case, such as $A.B + A.B.C = A.B$. The following pair of Venn diagrams shows that this is true.



ALISON FENTON

You may like to try writing down the duals of the other five special relations and draw Venn diagrams to confirm their validity.

Look again at the Venn diagrams given earlier. Comparing diagrams 5 and 8 shows us that the following important relation is always true: $\overline{A+B} = \overline{A}.\overline{B}$. Comparing diagrams 6 and 7 shows us that: $\overline{A.B} = \overline{A} + \overline{B}$. These two relationships are known as *de Morgan's Laws* and may be applied in more complex cases such as the case of three

variables ($\overline{A+B+C} = \overline{A}.\overline{B}.\overline{C}$ *and* $\overline{A.B.C} = \overline{A} + \overline{B} + \overline{C}$). De Morgan's laws may also be applied in stages:

$(\overline{A+B}).\overline{C}$
$= \overline{A}.\overline{B}.\overline{C}$ (using de Morgan's Law on the brackets)
$= \overline{A+B+C}$ (recombining using de Morgan's Law)

In common with normal algebra there are three further rules that may be applied to Boolean algebra. The *associative law* allows brackets to be moved:

$(A.B).C = A.(B.C) = A.B.C$
$(A+B)+C = A+(B+C) = A+B+C$

The order in which the letters are written may be changed according to the *commutative law* :

$A.B = B.A$
$A+B = B+A$

The *distributive law* allows brackets to be multiplied out:

$A.(B+C) = A.B + A.C$

EXAMPLES OF SIMPLIFICATION

1) Simplify $(\overline{A+B} + \overline{A}.B).B$
$= (\overline{A}.\overline{B} + \overline{A}.B).B$ (de Morgan)
$= \overline{A}.\overline{B}.B + \overline{A}.B.B.$ (distributive law)
$= 0 + \overline{A}.B$ ($\overline{B}.B = 0$, $B.B = B$)
$= \overline{A}.B$

2) Simplify $\overline{A}.\overline{B} + \overline{A}.B + A.B$
$= \overline{A}.(\overline{B} + B) + A.B$ (distributive law)
$= \overline{A} + A.B$ ($\overline{B} + B = 1$)
$= \overline{A} + B$ (dual of relation 6)

3) Simplify $\overline{\overline{A} + B} + \overline{\overline{A} + \overline{B}} + \overline{A}.B$
$= \overline{A}.\overline{B} + \overline{A}.\overline{B} + \overline{A}.B$ (de Morgan)
$= A.\overline{B} + A.B + \overline{A}.B$ ($\overline{A} = A$)
$= A.(\overline{B}+B) + \overline{A}.B$ (distributive law)
$= A + \overline{A}.B$ ($\overline{B} + B = 1$)
$= A + B$ (dual of relation 6)

A SIMPLIFIED XOR GATE

In the previous instalment we looked at an unsimplified circuit for an Exclusive-OR gate. Let us now examine the same problem again, but this time armed with the ability to simplify the Boolean expression and hence the circuit. The truth table for the Exclusive-OR gate is:

INPUT		OUTPUT
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

From the truth table we have previously decided that $C = \overline{A}.B + A.\overline{B}$. There is little simplification that can be made here and a five gate circuit would be required to implement this expression. However, there is an alternative way of approaching the problem. From the truth table, C can be said to be 1 if A and B aren't both 1 or both 0. In Boolean terms we can write down an alternative expression for C:

$C = \overline{A.B + \overline{A.B}}$

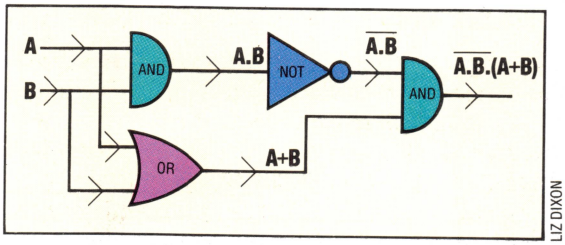
Using de Morgan's Laws repeatedly we can simplify the circuit to get:

$C = (\overline{A.B}).(\overline{\overline{A.B}})$

and, finally:

$C = \overline{A.B}.(A+B)$

This expression requires only four gates:



LIZ DIXON

A FULL ADDER CIRCUIT

Previously, we looked at the process of binary addition and designed a simple circuit to add two bits together and produce two outputs for the sum and carry digits in the answer. This circuit we called a half adder. If we call the first input X and the second input Y, we can then verify from the truth table for a half adder (see page 33) that the sum (or answer) output (S) can be represented by the expression: $S = \overline{X}.Y + X.\overline{Y}$. Using de Morgan's law this expression simplifies to: $S = \overline{X.Y}.(X+Y)$. The carry output (C) is simply: $C = X.Y$.

In binary arithmetic there are, in fact, three digits to be added in any one column of the addition sum. As well as the two digits to be added there is also a carry over from the previous column to be included. To be able to reproduce the process of binary addition we must design a circuit with three inputs and two outputs. If we call the carry from the previous column P then the truth table for a full adder will be:

	INPUTS		OUTPUTS	
P	X	Y	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Taking the cases when S = 1, an expression for S can be formed from the truth table:

$$S = \overline{P}.\overline{X}.Y + \overline{P}.X.\overline{Y} + P.\overline{X}.\overline{Y} + P.X.Y$$

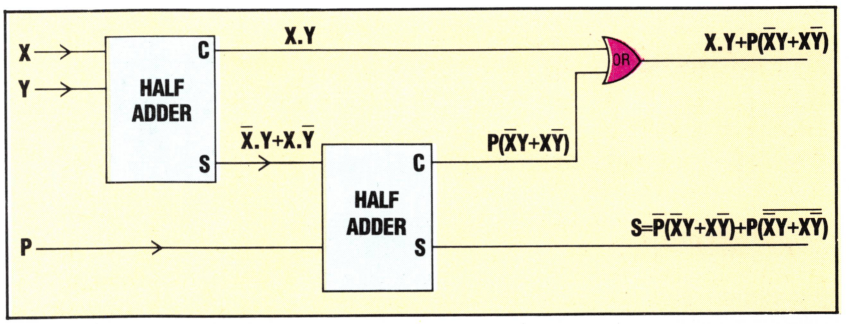
Using the rules we have learned we can simplify this expression:

$$S = \overline{P}.(\overline{X}.Y + X.\overline{Y}) + P.(\overline{X}.\overline{Y} + X.Y)$$
(distributive law)
$$S = \overline{P}.(\overline{X}.Y + X.\overline{Y}) + P.\overline{(\overline{X}.Y + X.\overline{Y})}$$
(de Morgan)

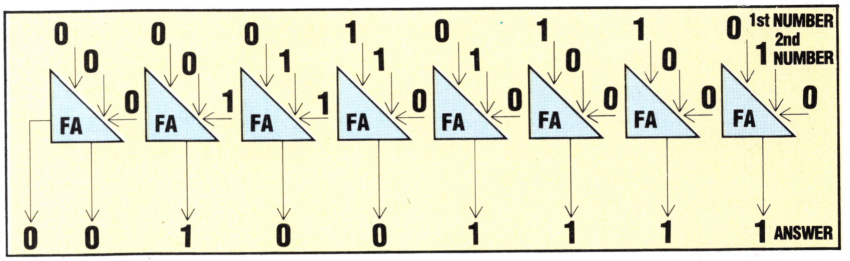
Similarly, we can form an expression for C. From the truth table:

$$C = \overline{P}.X.Y + P.\overline{X}.Y + P.X.\overline{Y} + P.X.Y$$
$$C = X.Y.(\overline{P} + P) + P.(\overline{X}.Y + X.\overline{Y})$$
(distributive law)
$$C = X.Y + P.(\overline{X}.Y + X.\overline{Y})$$
$(\overline{P} + P = 1)$

Note that $\overline{X}.Y + X.\overline{Y}$ is the sum output from a half adder circuit. Thus, a full adder circuit can be designed from two half adders.



The following example shows how a series of eight full adders combine inside the ALU to perform the binary addition of two eight-bit numbers.



EXERCISE 3

1) Simplify these expressions:

a) $A.(\overline{A} + \overline{B})$

b) $X + Y.(X + Y) + X.(\overline{X} + Y)$

c) $P.Q + \overline{P}.Q + \overline{P}.\overline{Q}$

d) $\overline{X + Y.Z + \overline{Z}.Y}$

2) A car alarm has an on/off switch and switches on the two front doors. The alarm will sound if either or both doors are opened when the on/off switch is set to on. Draw a truth table showing the three inputs (door A, door B, and the on/off switch) and the alarm output. Use your truth table to write a Boolean expression for the alarm sounding and draw a logic circuit for the alarm system.

3) A hall light is operated from a switch at the door, a switch at the bottom of the stairs or one at the top of the stairs. Design a suitable logic circuit.

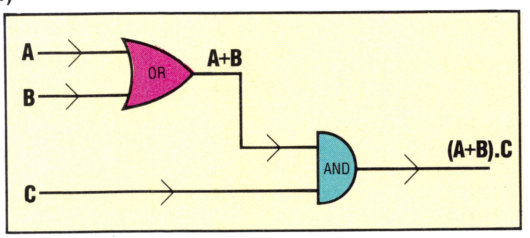
4) You are marooned on a desert island with two other people. One of these people always speaks the truth, the other always tells lies. For your own survival it is imperative that you find out which one will tell the truth. There are a number of questions that you could ask one of them to determine his or her identity. Draw up truth tables to investigate the possible answers. Here is an example to start you off.

'Do you always tell the truth?'

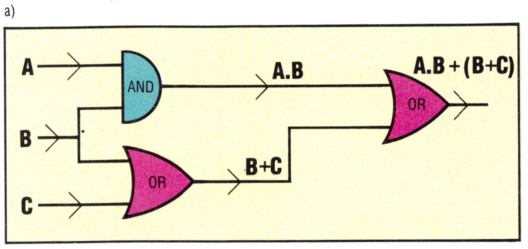
		POSSIBLE ANSWERS	
		YES	NO
POSSIBLE IDENTITY OF RESPONDENT	LIAR	1	0
	TRUTH-TELLER	1	0

Answers to Exercise 2 on page 33

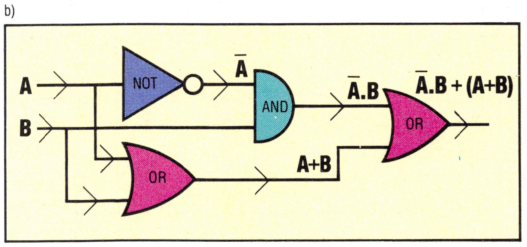
1)



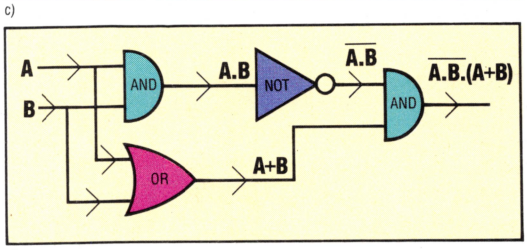
a)



b)



c)



d)

2a) $C = A.B$

b) $S = \overline{A.B}.(A + B)$

3a) $X = (A + B).(B.C + C)$

b) $X = \overline{A.B} + \overline{B}$

A

ALGOL

Nowadays, the home computer user has available a reasonably large range of programming languages, but in the days when computing was restricted to mainframes, there were really only three universal languages: FORTRAN, ALGOL and COBOL. While the first was really for engineers, and the last for business applications, it was ALGOL (*ALGO*rithmic *L*anguage) that was favoured by most scientists, mathematicians, and those who studied computer science. Indeed, ALGOL remains one of the most popular languages in universities. However, it has not proved popular with microcomputer users and is available on only a limited number of low-cost machines, perhaps the best example being the Research Machines 380Z.

ALGOL's strengths lie in its ability to handle procedures, making it a highly structured language like PASCAL; and it also features a large vocabulary of pre-defined scientific and mathematical functions. Oddly, the original specification of the language made no provision for input or output (there was no equivalent to BASIC's INPUT and PRINT), with the result that programs had to be extensively re-written from machine to machine.

ALGORITHM

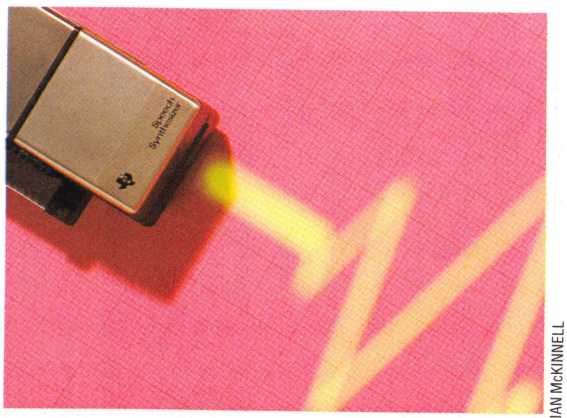
An algorithm is a method or procedure for solving a particular problem. Algorithms therefore exist outside the world of computing, but their main application is now in programming. Sometimes, writing a program simply consists of expressing a well-known algorithm (to find the square root of a number, for example) in the required programming language. More often, however, the greater part of writing a program consists of developing an algorithm for solving a new type of problem. What is the algorithm, for example, to find your way out of a maze, or to create a realistic-looking descent of an aircraft onto a runway?

With many mathematical and statistical problems, finding an algorithm is easy, but it is important to find the *best* algorithm. Two algorithms may produce the same result, but one may be more economical in memory usage, while the other is faster. One classic example is the inversion of a matrix, essentially the extensive manipulation of the data in a numeric array variable. Students learn about matrices using small arrays (say 3 × 3) and for which there is a particularly easy algorithm. But the same algorithm applied to a 15 × 15 array (not large in mathematical terms) would take something like 15 million years to complete, using the fastest available hardware! Yet a different algorithm could finish the task in a matter of seconds.

ALLOPHONES

Allophones are concerned with verbal communication and, hence, are important in the field of speech synthesis. The single sound elements that help us distinguish one word from another are called phonemes. In English, for example, the words 'pat' and 'mat' are differentiated by the phonemes 'p' and 'm'. Phonemes occur in subtly different variants called allophones; thus the slightly different vowel sounds in 'pat' and 'pad' are allophones of the phoneme 'a'. Speech generated from allophones sounds artificial but is relatively easy to program. To make things even easier, some speech synthesis units come with a program that will accept plain English from the keyboard and then translate it into the allophones required. It will, though, sometimes make mistakes on a word like 'row', which has two separate pronunciations very dependent on context. The method of speech synthesis that makes use of phonemes and allophones is known as synthesis by rule.

The other method is synthesis by analysis, in which the computer digitises words spoken by a human into a microphone, analyses the data for frequency and other characteristics, and then stores the words in condensed form. When the words are recreated by the reverse process, the resulting quality is very high, and it is even possible to identify the original speaker. The drawback is that the size of the vocabulary is limited by the memory capacity.



IAN McKINNELL

ALPHANUMERIC

Alpha means letters of the alphabet, and numeric refers to the digits 0 to 9, so alphanumeric is really a technical way of saying 'letters and numbers'. In practice, the term also covers punctuation signs and some of the more abstract symbols found on a computer's keyboard. What the term does not cover is graphics symbols and the various non-printable characters that some computers feature, such as 'carriage return', 'sound bell' and 'clear screen'. The word alphanumeric used in a description of the operation of a software package may be a special feature or it may be a severe restriction. A stock control program that allowed alphanumeric codes for each product (as distinct from purely numeric) would fall into the former category, while a printer with only an alphanumeric character set would mean that your program listings wouldn't include any graphics characters.

TO THE LIMIT

Few computer manufacturers expect to sell a million units of any one machine; yet Sinclair sold a million Spectrums in little more than 15 months. Now in its third version, the Spectrum still has some idiosyncratic features, and a few that are sub-standard, amongst them its keyboard, and its limited capacity for expansion.

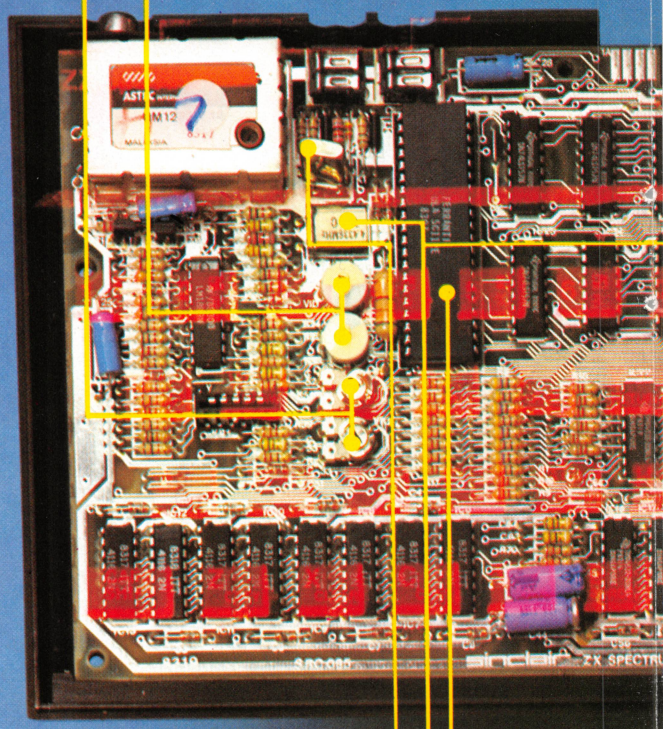
As a piece of hardware, the Spectrum's most noticeable feature is the keyboard. While it is true to say that the configuration is QWERTY, that is as far as its resemblance to a typewriter goes. Each of the 40 keys is part of a membrane that allows the keys a certain amount of travel: the 'feel' of the keys is yielding and 'spongy'.

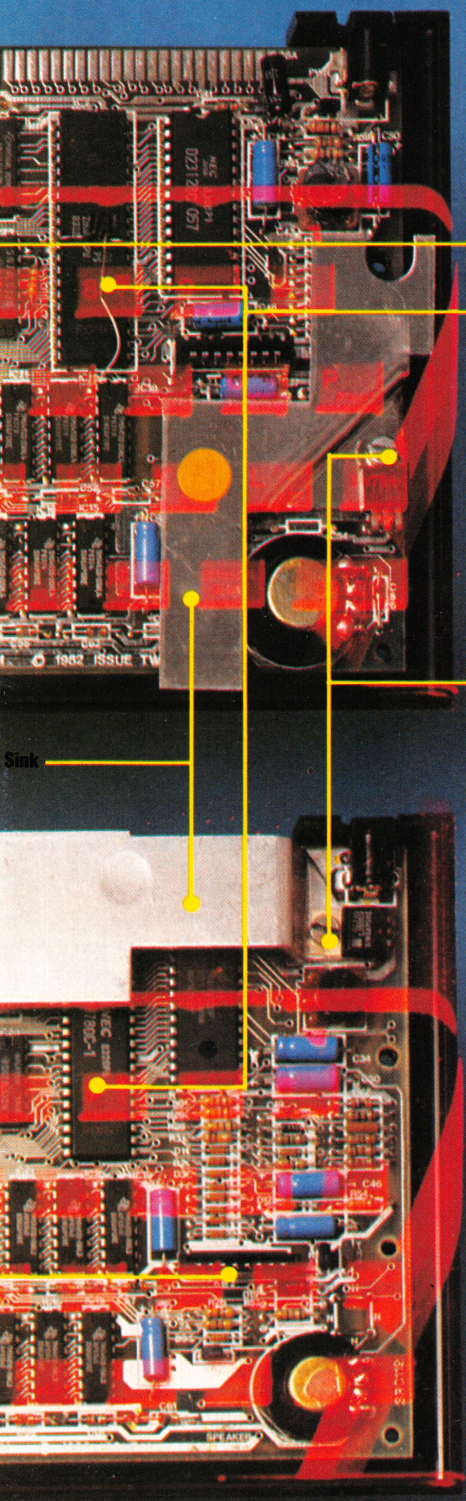
The Spectrum has a system bus connector, which allows the user to connect the ZX Printer (originally designed for the ZX81), the ZX Interface 1 and the ZX Interface 2 — or all three peripherals at once. There are also MIC and EAR sockets, which allow cassette storage of programs. The program loading procedure is not very satisfactory. Although successful loading and saving is indicated by reassuring blue and yellow strips in the border area of the screen, in order to save a program you must first disconnect the EAR lead. Equally annoying is the lack of a Reset button on the computer, which means that whenever a system crash is experienced the power lead must be pulled out — which could eventually weaken the connections. Fortunately, a number of small independent companies produce add-on devices that incorporate both a 'Save or Load' switch and a Reset switch.

This is only one example of the way in which Sinclair Research appears to have been beaten to the mark in supporting its machine. This seems to be deliberate company policy, however, for while enjoying the spoils of its computer sales Sinclair can move on to another project, such as the QL. Even so the company has developed the ZX Microdrives and the associated ZX Interface 1 unit, which provided the back-up storage potential of 680 Kbytes, an RS232 port and the concept of networking up to 64 Spectrums. There was also the introduction of the ZX Interface 2 unit, which allows access to ROM-based software and two joysticks to be connected.

Sinclair Research have also left the production of software to others: having honoured certain packages produced by independent software houses with its seal of approval. Its library of 50 tapes covers such diverse subjects as education, business, domestic, utilities and games.

Spectrum Evolution
Between its introduction in 1982, and its partial eclipse by the QL at the beginning of 1984, Sinclair's ZX Spectrum sold more than a million units, and went through three versions of its motherboard (the printed circuit board that holds all the major components). Version 1 was current for only the first 60,000 units sold, and so is not common. Versions 2 and 3 differ in two major respects. Firstly, one could 'fine tune' the video output circuitry in Version 2 by means of the two trimming capacitors and the two variable resistors shown. Secondly, the 'temporary modification' to the Version 2 microprocessor had been properly executed by the time Version 3 was introduced. The heat sink is in a different place because the voltage regulator chip has been relocated closer to the power input socket





Colour Video Clock Crystal
Runs at 4.4336 MHz

Z80A Central Processor Unit
Note the 'temporary modification' to Version 2 motherboards, bridging pins 11 and 30 together with a transistor. The effect of this modification is to allow the ULA to be selected only when address line zero and the IORQ (Input/Output Request) are both low together

Voltage Regulator

Sink

IAN McKINNELL

SINCLAIR SPECTRUM

PRICE
£99.95 (16K version)
£129.95 (48K version)

DIMENSIONS
233 × 144 × 30mm

CPU
Z80A

MEMORY CAPACITY AND SPEED
16K and 48K
3.5 MHz

SCREEN CHARACTERISTICS
The screen is divided into 24 lines of 32 characters. Bit-mapped graphics with a 256 × 192 resolution. 16 pre-programmed block graphics characters plus 21 user-definable graphics characters. Eight colours plus flash and two brightness levels. Independent border colour

INTERFACES AND PORTS
System bus connector. Sockets for cassette storage and TV

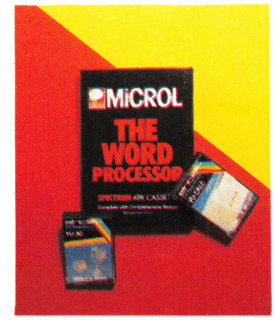
AVAILABLE LANGUAGES
BASIC and Z80 Assembly language. Further expansion via software allows FORTH, LOGO, Micro-PROLOG and many others to be used

KEYBOARD
40 moving-key membrane ASCII keyboard. The design allows up to eight functions to be accessed from one key via a series of Shift keys

DOCUMENTATION
The computer comes complete with an introductory manual and a main manual containing a guide to the micro's operation as well as advice on Sinclair BASIC

STRENGTHS
At £130 for a 48K version, the Spectrum offers an ideal opportunity for the newcomer to experiment. The Spectrum is also blessed with much greater software and hardware support from independent producers than any other computer

WEAKNESS
The keyboard, although cleverly designed, does not offer typewriter-like facilities. The screen display is configured in a highly non-standard way, which may cause problems for the inexperienced



The Spectrum File
In addition to the vast number of games packages available for the Spectrum, there is also a very respectable body of business and household software on the market, much of it priced at less than £10. With the introduction of the ZX Microdrive, there is no reason at all why the Spectrum cannot maintain large databases, spreadsheets and word processing applications.

Psion, one of Sinclair's closest collaborators in the field of software production, has been responsible for originating much of this material. Most notably it has produced the successful VU-series of packages, including VU-CALC, VU-File and VU-3D. And despite the deficiencies of the Spectrum's keyboard, Microl have produced The Word Processor, which offers storage space for up to 10 A4 pages of text, and most of the facilities normally found in more complete text editing packages (including the ability to merge files)

COMMANDING MOVES

Commodore have for many years produced a range of 5¼in floppy disk drives. All Commodore drives are 'intelligent', containing their own microprocessor and associated RAM. The Commodore disk operating system (DOS) is available in several broadly similar forms that are resident in ROM within the drives.

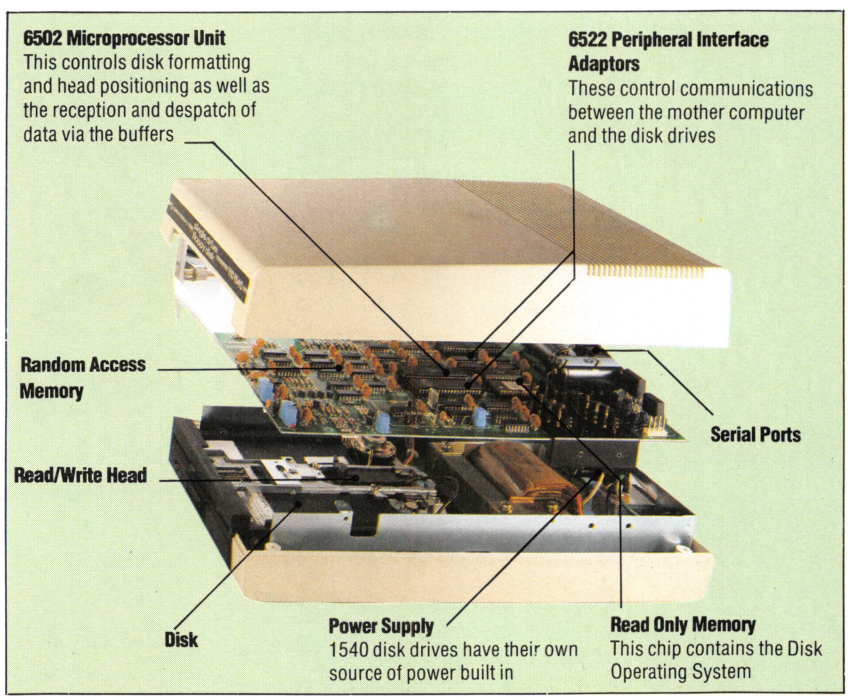
The main problem with intelligent disk drives is that they are expensive to manufacture. After the introduction of the Vic-20 home computer, Commodore launched an inexpensive single drive version of its successful PET drives called the Vic-1540. The Commodore 64 incorporates similar facilities to the Vic-20 for accessing the 1540, but minor differences made it necessary to carry out a POKE before using the drive and a further POKE on completion. This tiresome process is no longer necessary, however, as Commodore made changes to the DOS to rectify the defect and re-launched the 1540 as the 1541. This newer version is fully compatible with both the Vic-20 and the 64. For simplicity we will refer to both drives as the 1541, as there is no difference in the way they are used.

The 1541 is controlled via a 6502 microprocessor, two 6522 Versatile Interface Adapters (VIAs), two Kbytes of RAM and eight Kbytes of ROM, which contains the DOS. The DOS supplied is very powerful and enables complex routines to be programmed to create and manipulate program (PRG) files, sequential data (SEQ) files and random access files, all with sophisticated error-checking procedures. Computer control is exercised via a serial version of the IEEE488 interface. This interface supports the same commands as its more powerful parallel equivalent — through which the other Commodore peripherals are controlled — and allows serial IEEE488-fitted peripherals to be 'daisy chained' so that, for example, a disk drive can output files to a printer while the computer carries out another task. This is accomplished by the use of commands with associated logical file numbers and device numbers.

Diskettes are formatted into 35 tracks on a single side, each track being arranged in sectors, with 21 in the outermost track down to 17 in the innermost. Each sector contains one 256-byte block of file data plus timing, identification and checksum data. Each diskette stores 683 blocks, of which 664 are available to the user. This gives a maximum capacity of approximately 170 Kbytes, dependent on what type of files are stored. DOS manages the distribution of data on a diskette by maintaining a Block Availability Map (BAM) and Directory. The BAM is stored in track 18, sector 0 and consists of 144 bytes that signify which blocks are in use and which are free for storage. The Directory starts at sector 1, track 18 and is a list of a maximum of 144 files, by file name, that contains specific information relating to file type and how many blocks it consists of. Both the BAM and Directory are updated as data is written to or removed from the diskette.

Despite a high purchase price, the flexibility of the Commodore intelligent disk drive system gives value for money. The scope offered by its reliable mass storage (which can be incorporated in a peripheral management system and does not encroach on computer memory or processor time) justifies the expense. However, it is unfortunate that, because of the serial interface, the 1541 is comparatively slow in operation. Commodore do not even quote a data transfer rate or average access time in the user manual supplied. Although it is in excess of 50 times faster than cassette storage it is likely that the 1541 has the longest access time of all the popular disk systems. If a large storage capacity is required, it is possible to purchase an interface that connects to the serial port and emulates the parallel port fitted to PET computers. This does not speed operations up particularly, but it does enable you to connect the full range of PET peripherals to a Vic-20 or Commodore 64.

Commodore Disk Drive
Available for both the Vic-20 and the Commodore 64, and a standard feature of the later SX-64 portable computer, Commodore's 1540 disk drive is an intelligent unit that makes no demands on the host computer's CPU or memory. In fact, it has a processor of its own — the same MOS Technology 6502 that powers the computers themselves



IAN McKINNELL

Commodore Disk Commands

Commodore DOS supports a wide range of commands designed to enable the user to construct complex random access file handling programs, as well as the usual program and data file handling procedures. These commands are used as shown below. In each case 8 is the identifying device number of the disk drive.

SAVE

Creates named (up to 16 characters) program (PRG) files that can be programs or sequential data. The format is as follows:

SAVE"FILENAME",8

LOAD

Constructed as:

LOAD"FILENAME",8

This copies the specified PRG file into RAM from the bottom of user memory upwards. The command

LOAD"FILENAME",8,1

copies the specified file back into the memory locations from which it was originally SAVEd.

LOAD"$",8

copies the disk directory into user memory. It can then be LISTed like a BASIC program and contains the following:

Disk name
2 character disk identifier
Up to 144 file names
Type (PRG or SEQ) for each file
Length of each file in blocks
Number of free blocks available

VERIFY

Constructed as:

VERIFY"FILENAME",8

compares the specified file with the file currently contained in user memory and generates an error message if they are different. Used to check if files have been SAVEd correctly.

OPEN

Establishes a unique communication channel that is identified with a 'logical file number' (LFN) within the range 1 to 255. Up to 10 LFNs may be open at one time. OPEN also establishes a 'secondary address' (SA), which determines the way the device accessed will behave. The only disk drive secondary address is 15, which accesses the priority 'command' channel. OPEN is entered as:

OPEN LFN,8,SA

CLOSE

Takes the format:

CLOSE LFN

Terminates the specified logical file. Logical files should always be CLOSEd when they are no longer required.

PRINT#, INPUT# AND GET#

PRINT# operates in a similar manner to PRINT except that data is output, as a SEQ file, to the specified OPEN logical file instead of the screen. Constructed as:

PRINT#LFN,"DATA" or
PRINT#LFN, A$,B$,...

In the same way, INPUT# and GET# read SEQ files. INPUT# retrieves string data but is only effective if the stored strings are separated by semi-colons or commas, otherwise INPUT# will treat the data as one long string. GET# retrieves data one byte at a time, including semi-colons and commas. This is most useful if the contents of a file are unknown and not separated. The following are examples of the command formats:

INPUT#LFN,A$,B$...
GET#LFN,A$,B$...

When PRINT# is used in conjunction with a logical file OPENed to the command channel (e.g. OPENLFN,8,15) like this:

PRINT#LFN,8,15,"command string"

it is transformed into the most powerful disk handling command available. Command strings are used to implement disk maintenance commands and advanced random access (relative — REL) file commands.

Disk Maintenance Commands

When used in conjunction with PRINT# or OPEN on the command channel, in the format given, these command strings perform these functions:

NEW

Formats and names the diskette
Constructs BAM and Directory
Assigns the 2 character disk identifier (DI)
Command:"N:DISKNAME,DI"

INITIALISE

Checks BAM in disk RAM with BAM on disk
Command:"I"

VALIDATE

Deletes blocks allocated by advanced REL commands not held on directory and files not CLOSEd after they were written
Writes a new BAM
Command:"V"

RENAME

Changes the Directory listing of a specified file
Command:"R:NEWNAME=OLDNAME"

SCRATCH

Deletes specified files from disk and directory
"S:FILENAME 1,FILENAME 2,..."

COPY

Writes a copy of a file on the same disk
Command:"C:DUPNAME+ORIGNAME"
Joins SEQ files and writes them as a single SEQ file on the same disk. Known as 'concatenating'
Command:"C:CONNAME=NAME1, NAME2,..."

Error Checking

The front panel of the 1541 disk drive holds a green 'power on' LED and a red disk condition indicating LED where:

On = Reading from or writing to disk
Off = Waiting for instructions
Flashing = DOS has detected an error

To find out the nature of the error it is necessary to read the DOS error channel. The following program prints the error codes generated by DOS. A list of error codes and their meanings is given in the disk drive user manual.

```
10 REM**DISK ERROR
   CHECK**
20 OPEN15,8,15
30 INPUT#15,EN,EM$,ET,
   ES
40 PRINTCHR$(147)
50 PRINT"ERROR NO"EN
60 PRINTEM$
70 PRINT"TRACK"ET
80 PRINT"SECTOR"ES
90 CLOSE15:END
```

THE ABC OF BBC

We continue our appraisal of the built-in BASIC of the most popular home computers by looking at BBC BASIC. This dialect is as full of fascinating abilities and facilities as the machine itself; and just as the BBC Micro marked a new phase in the design of home computers, so BBC BASIC is considered amongst the best of the dialects.

BETWEEN THE LINES

Don't forget to number your program lines in multiples of ten. Even the best programmers have to make insertions into their programs from time to time....

The criticism most often made of BASIC is that it is an unstructured language that encourages (or at least does nothing to check) bad programming habits in the beginner, especially the 'quick and dirty' approach to problem solving, which leads, for example, to undisciplined use of GOTO.

The use of ELSE with the IF..THEN statement can eliminate the commonest use of GOTO, by permitting both true and false cases of a condition to be dealt with in the same statement. For example, these lines:

```
1500 IF TEST >0 THEN GOTO 1800
1600 PRINT "VALUE OUT OF RANGE"
1700 GOTO 1900
1800 PRINT "NO PROBLEM"
1900 NEXT L
```

can be replaced by:

```
1500 IF TEST>0 THEN PRINT "NO PROBLEM"
     ELSE PRINT "VALUE OUT OF RANGE"
1900 NEXT L
```

GOSUB usually takes a line number as its argument, which has two disadvantages — first, GOSUB 1000, for example, gives no clue as to the purpose of the subroutine at line 1000; and second, specifying line numbers makes the program very difficult to renumber or merge.

GOSUB, like GOTO, is relatively slow in execution because the specified line has to be searched for in the program every time the instruction is obeyed.

BBC BASIC's functions and procedures answer these objections. Both are subroutine-like blocks of code, but are called by name rather than line number, so they can be self-documenting, or at least meaningful in the list, and need not be affected by subsequent renumbering or merging. Furthermore, function and procedure calls are generally executed more quickly than the GOSUB and GOTO commands.

Procedures and functions begin with DEF PROC or DEF FN, followed by a name, and usually (but not necessarily) a parameter list. For example:

```
1200 DEF FNcalc(a,b,c)=(a-b)*c/100 and
2500 DEF PROCoperate(w,x$,y$,z)
```

The definition will use these parameters as if they were program variables. When the program calls the function or procedure, however, the parameters, or dummy variables, may be replaced by any variable number or literal expression of the same data type as the original parameter. For example:

```
250 result=FNcalc (price,cost,12)or
545 PROCoperate (6, names$, "smith",array(12))
```

The values of the parameters are then used in place of the dummy variables in the definition. Notice that a function can be used in an expression as if it were a variable or arithmetic quantity, whereas a procedure call must be used as if it were a BASIC command. The LOCAL command, which defines variables for use exclusively inside the definition block, removes the chance of a common subroutine bug. For example, consider this code:

```
100 FOR K=1 TO 10:GOSUB 500:NEXT K:END
500 FOR K=1 TO 5:PRINT"****":NEXT:RETURN
```

Here the variable K is used as the loop counter in the main program line 100, and again in the subroutine in line 500 — an oversight that will seriously affect execution, but which can be extremely difficult to avoid (or to trace in a long program). In a BBC procedure, however, this danger is avoidable:

```
100 FOR K=1 TO 10:PROCstars:NEXT:END
500 DEF PROCstars
520 LOCAL K
540 FOR K=1 TO 5:PRINT"****":NEXT
560 ENDPROC
```

The LOCAL command means that between lines 500 and 560 the variable K is a new variable, independent of the variable K anywhere else in the program, and having no effect on the value of K elsewhere. (Notice that PROCstars is a procedure without parameters.)

REPEAT..UNTIL is a loop structure in which iteration continues until the conditional expression that follows the word UNTIL is true; control then passes to the statement after UNTIL. For example:

```
200 DATA 12,234,31,45,65,0,76,81
250 REPEAT
300 READ number:sum=sum+number
350 UNTIL number=0
400 PRINT "Sum is ";sum
```

This is much more readable, and is much less prone to error than either a GOTO loop or a dummy FOR..NEXT loop.

BBC BASIC has the useful debugging aids TRACE, ON ERROR..., and ERL. TRACE causes program line numbers to be displayed on the screen as they are executed; ON ERROR GOTO (or GOSUB), means that any normally fatal error (including pressing the ESCAPE key) during program execution will cause control to pass to a user-defined error-handling routine (such as a dump of all variable values). ERL is a system variable containing the number of the line on which an error occurred.

The BBC has a wealth of unique extensions to BASIC, such as the Operating System calls, the VDU command, the various system variables and the Assembler, which are themselves sufficiently complex to be the subjects of individual articles later in the course.

BBC Graphics

The BBC BASIC commands that relate directly to graphics are:

MODE

This selects the computer display mode with MODE N where N = 0 to 7:

Mode	Graphics	Colours	Text
0	640×256	2	80×32
1	320×256	4	40×32
2	160×256	16	20×32
3	text only	2	80×25
4	320×256	2	40×32
5	160×256	4	20×32
6	text only	2	40×25
7	Teletext		40×25

BBC Model A computers can access only modes 4, 5, 6 and 7; Model B has access to all modes. In modes 0 to 6, the character set can be changed by the user with the VDU command. Mode 7 Teletext characters are fixed and don't correspond to standard ASCII code.

COLOUR

This sets one of 16 colours for text and background according to the mode selected with:

COLOUR N

where N is 0 to 15 for text colours and 128 to 143 for background colours. The colours set by each value of N do not remain constant from mode to mode. Lists of N values related to colours for each mode are given in the user guide.

VDU

This is a highly useful command. VDU A is equivalent to PRINT CHR$(A). Similarly, VDU A,B,C has the same effect as PRINT CHR$(A); CHR$(B); CHR$(C);. This means that the many complex text and graphics routines under the control of the 32 CHR$ codes, duplicating the effects of most of the graphics-related BASIC commands, can be constructed with a small number of VDU commands.

CLG

This clears the graphics area of the current screen and moves the cursor to its 'home' position at the bottom left of the screen.

CLS

This clears the text area of the screen and moves the text cursor to its 'home' position at top left. Any graphics on the screen will also be cleared.

DRAW

This draws lines on the screen in modes 0, 1, 2, 4 and 5. Constructed as:

```
DRAW X,Y
```

The point defined by the X and Y co-ordinates is the end of the line. The starting point can be either the end point of the last line drawn, or a point defined by a MOVE command.

GCOL

This sets the current graphics foreground and background colours with:

```
GCOL N,M
```

where N sets how colour is to be used (0 to 4) and M defines the logical colour using the same principles as COLOUR. N has the following effects:

0 – Plot the colour specified by M
1 – OR M colour with present colour
2 – AND M colour with present colour
3 – Exclusive–OR M colour with present colour
4 – Invert present colour

MOVE

This positions the graphics cursor to a specified point with:

```
MOVE X,Y
```

It has the same effect as DRAW but without drawing a line.

PLOT

This can be used for many graphic functions including point, line and triangle drawing. Constructed as:

```
PLOT K,X,Y
```

where K defines the type of graphic PLOTted. K takes values in the range 0 to 225 to specify the type of lines to be drawn and the colours they take according to lists given in the user guide.

POINT

This gives the number relating to the logical screen colour of the specified screen co-ordinate with:

```
NUMVAR = POINT(X,Y)
```

where NUMVAR is a numeric variable.

MEMORY MONITOR

The hexadecimal number system appears to be a complicated and cumbersome alternative to our everyday decimal, but it is in fact an extremely useful and easily understood way of dealing with memory addresses and their contents when faced with the limitations of an eight-bit byte system of memory.

At this point in the Machine Code course, it's worth returning to the question of number representation. We are already familiar with the decimal (or denary) number system which we use most of the time, and we've investigated the binary system (see page 18). It is well to remember that both the decimal and binary systems are simply alternative expressions of the same concept — number. Most human beings, for example, have the same number of fingers per hand. You may say that the number is five, and someone else may call it *fünf*, or *cinq*, or *pente*; but a moment's empirical investigation will show that you're all talking about the same quantity or number — it is only your representation systems that are different. Different but equivalent. There is a one-to-one correspondence between all numbers expressed in English and all numbers expressed in any other language, and there is internal consistency in all these systems. Arithmetic yields the same results irrespective of the language used to describe the individual components of an arithmetic expression.

RUN TIME

One Edinburgh branch of a national chainstore sold 75 48 K Spectrums in less than 15 minutes just prior to Christmas 1983 — only a few days after Sinclair Research had celebrated the production of the millionth unit of that model

Different number systems are exactly similar to different languages. The number of fingers on a normal human hand is not changed by being called *fünf* or five, neither is it altered by being written 5 or 101b (the b here showing that 101 is to be interpreted as a binary number). The only reasons for choosing one system or the other are either custom or convenience.

We find it convenient to use decimal representation at first because it is the number system most commonly used around us. But it is not the only system. Digital clocks, for example, use a bizarre system of arithmetic: part decimal, part modulo 60 (there are 60 minutes in an hour and 60 seconds in a minute), and part modulo 24 (24 hours in a day). Before 1971 British money was reckoned in units of 12 (pence in a shilling) and 20 (shillings in a pound). Learning to use this system took years of agonised schooling — how many people ever really learned how to express shillings and pence as decimal fractions of a pound?

When talking about computers, we find it instructive to begin by talking about binary numbers because they so closely model the computer's electrical operations, being simply sequences of on-off switch states. If we only ever wanted to talk about single-byte numbers, then binary might serve as a complete alternative to decimal — translating eight-bit binary into decimal becomes surprisingly easy with a little practice. Unfortunately, memory addresses in particular and useful numbers in general are usually too large to be fitted into one byte, so computer programmers and engineers over the years have felt the need for a number system with the logical convenience of binary, as well as the range of decimal. Two systems have been used for these reasons: hexadecimal and octal. The first, now standard in microcomputing, is usually called *hex*, and is based on the number 16. Octal, based on 8, has been widely used in mainframe computing, but is increasingly being replaced by the hex system.

USING HEX NUMBERS

In looking at decimal and binary representation, we have seen two consequences follow from the choice of number base: the base is the number of unique digits needed in the system, and it is the multiplicative factor in the positional notation. For instance, there are ten unique digits in decimal (0–9), and the value of a decimal digit is multiplied by ten each time it shifts leftwards in a decimal number.

Hexadecimal, therefore, requires 16 unique digits, and they are the digits 0–9 and the letters A–F. Counting in hex is simply a matter of working through the single digits and then re-using them in positional notation. The hex number after 9, therefore, is A (decimal 10); next is B; next C; and so on until F (decimal 15). That exhausts the single digits, so the hex number after F is 10 (say this as: 'one-zero hex'), which corresponds to 16 decimal. From this we can see how the single digits are used, and that the value of the columns in a multi-digit hex number increases by a factor of 16 with leftward movement. In a decimal number we call the columns: Units, Tens, Hundreds and Thousands. By comparison, in a hex number the columns are: Units, Sixteens, Two-Hundred-and-Fifty-Sixes and Four-Thousand-and-Ninety-Sixes. By comparing the changes in the binary column with the changes in the hex column, you should be able to see the major advantage of hex numbers: the range of a four-bit binary number is exactly that of a single-digit hex number (i.e. 0 to 15 decimal). Some examples should make this clear:

Decimal	Binary	Hex
0	00000000	0
1	00000001	1
2	00000010	2
3	00000011	3
.....		
7	00000111	7
8	00001000	8
9	00001001	9
10	00001010	A
11	00001011	B
12	00001100	C
13	00001101	D
14	00001110	E
15	00001111	F
16	00010000	10
17	00010001	11
.....		
24	00011000	18
25	00011001	19
26	00011010	1A
27	00011011	1B
.....		
31	00011111	1F
32	00100000	20
33	00100001	21

The range of a single eight-bit byte number, therefore, is eight binary digits, or two hex digits:

```
      0 to 255        in decimal
00000000 to 11111111  in binary
      0 to FF         in hex
```

To convert a hex number into binary, therefore, you simply express each hex digit as a four-bit binary number. If a single-byte number is expressed as a two-digit hex number, then the leftmost hex digit corresponds to the four leftmost binary bits, while the rightmost hex digit corresponds to the four rightmost binary digits. Splitting a byte like this gives us two 'nybbles' (a nybble is half a byte). The leftmost nybble, corresponding to the leftmost hex digit, is called the upper or most significant nybble; and the rightmost nybble is called the lower or least significant nybble. Here is an example:

```
          Upper   Lower
          Nybble  Nybble
            ↓       ↓
206   =   1100    1110   =   C E
 ↑          ↑              ↑
decimal  binary equivalent   hex equivalent
```

It is important to make ourselves as familiar as possible with the hexadecimal number system, for the simple reason that it makes eight-bit byte manipulation much easier than if we were using binary. Convincing yourself of this requires a little practice, not just with number examples, but particularly with memory addresses and the contents of memory bytes. Once this becomes important — and very soon it will — you'll wonder how you ever managed in decimal.

We give programs in this instalment of the Machine Code course, for the BBC Micro, Commodore 64, and the Spectrum, that allow us to look at the contents of specified bytes in memory. These 'Mempeek' programs, as we have called them, ask you first to state the 'Start Address' (i.e. specify the first byte number) and then to give the number of bytes to be looked at. If, for example, you wished to specify byte1953 as your beginning point and request that the contents of the four following bytes be displayed, then the screen will show the decimal number 1953 in the leftmost column, and then list the contents of byte1953, byte1954, byte1955 and byte1956 in the next four columns.

Bear in mind that if the machine shows that byte1956 contains the decimal number 175, what we mean is that in one of the memory chips, an area that the machine calls byte1956 carries a pattern of eight voltage levels. If 0 volts is represented by 0, and 5 volts by 1, then byte1956 carries the voltage pattern 10101111. This we choose to interpret as a binary number, and its decimal equivalent is 175.

It is vital to remember that we use an imprecise kind of shorthand most of the time that we talk about computers; and expanding it into physical description is always salutary, and should help to avoid confusion.

The contents of a byte displayed on the screen are not the 'actual' contents. What we see are character data that have been assigned to the voltage patterns of the bytes. This means that having interpreted the voltage patterns as binary numbers, and having converted the binary to decimal numbers, we are going one step further and converting decimal numbers into characters according to ASCII — the American Standard Code for Information Interchange. This character data is displayed in the last column of the display. This is an internationally recognised code implemented in most computers, which substitutes decimal numbers between 0 and 127 for all the characters on a keyboard (historically, a teleprinter keyboard). In this code the decimal number 65 means the upper-case character 'A', 66 means 'B', 67 means 'C', and so on. Among the non-alphabetic characters, 32 means a space character, 42 means an asterisk, 13 means the Return key, and so on.

The printable ASCII characters start at number 32 and finish at number 127. Codes outside that range are undefined, or not printable, or specific to particular machines. Because of this, when we run the Mempeek programs, the monitor prints a dot to represent any byte containing a number out of range. In the next instalment of the course, we will provide a comprehensive ASCII character set for the values between 0 and 127.

An investigation of the ASCII character set is particularly useful as background to a full understanding of machine code for two important reasons. Firstly, it reinforces the point that how you interpret memory contents is entirely a matter of choice. You can say that a

byte contains a number, or an address, or a coded character, or an instruction, or whatever you like. In any case it will be data waiting to be interpreted. Secondly, it does give a rather more understandable view of memory, especially those parts of it which do actually contain character data, some of it used by the machine's Operating System, and some of it used by you.

Operating System data includes all the Error and Prompt messages — READY, for example, or NONSENSE IN BASIC, or START TAPE THEN PRESS RETURN — anything that it is capable of saying to the user has to be ASCII coded and stored in memory. You may never have thought of that, and it's a revealing insight into a computer's limitations as an 'intelligent' machine. Our intelligence is obviously different: we don't memorise messages like that, we simply frame a thought and then generate an appropriate combination of words to express it.

Memory Maps

A memory map is a simple schematic representation of the use to which memory is being put, showing the parameters of specific areas. Some areas of memory are always used for the same purpose. On the Commodore 64, for example, bytes 0 to 1024 are used by the BASIC Operating System as a Work Area. Other areas of memory have varied uses depending upon the program size and state. The boundaries between these areas can be either *fixed* (shown in our diagrams below as solid lines) or *floating* (shown as broken lines). Fixed boundaries never alter, while floating boundaries are for those areas of memory that fluctuate according to need. In the Commodore memory map, the Screen RAM boundaries are fixed (at bytes 1024 to 2048), while the boundaries of the memory area where the BASIC Variables are kept fluctuate according to how many variables are being used at any given time.

The Mempeek programs on page 59 can be used to locate the current positions of the floating boundaries in the memory of your machine. The Commodore has six floating boundary pointers (also called System Variables). An example is given in the panel below explaining how the contents of a pair of bytes are used to calculate the required memory address. BBC BASIC has four System Variables to determine, and the Spectrum has five.

It must be remembered that a memory map is a static representation of something that is forever changing while the machine is in use. Each of the floating boundaries is subject to change at any time. We give ideas, on the next page, of how you can extend the Mempeek programs to observe variations in the pointer values.

Commodore System Variables

43,44 Start of BASIC PROGRAM TEXT
45,46 Start of BASIC VARIABLES
47,48 Start of BASIC ARRAYS
49,50 End of BASIC ARRAYS
51,52 Bottom of BASIC STRING STORAGE
55,56 Top of BASIC STRING STORAGE

Example

Use the Mempeek program on page 59 to inspect the contents of these bytes. As an example, your screen display could look like this:

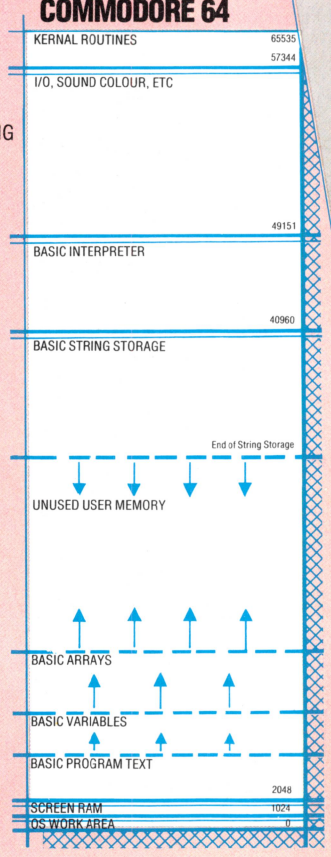
43 0 8 11 9

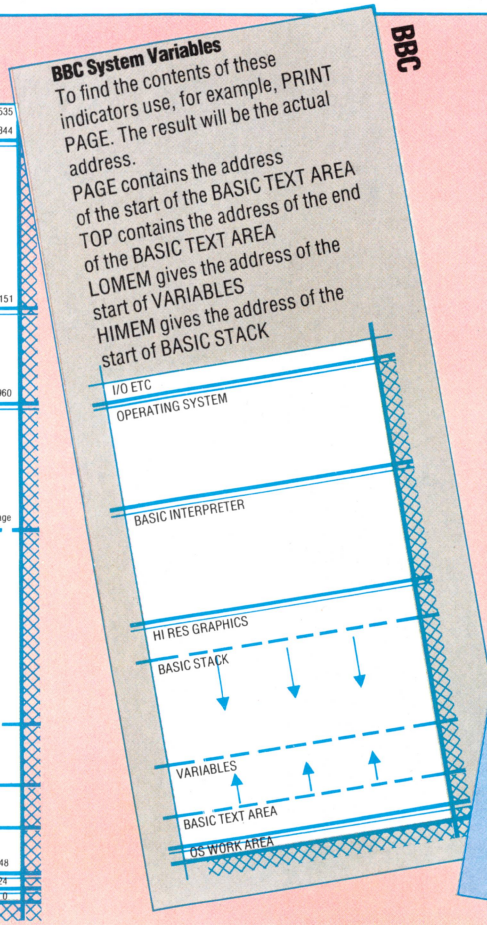
The first column is the address of the first byte accessed. The second and third columns display the contents of byte43 and byte44. These are the offset and page bytes (see page 36) of the address of the start of the BASIC Text Area. This is calculated as:

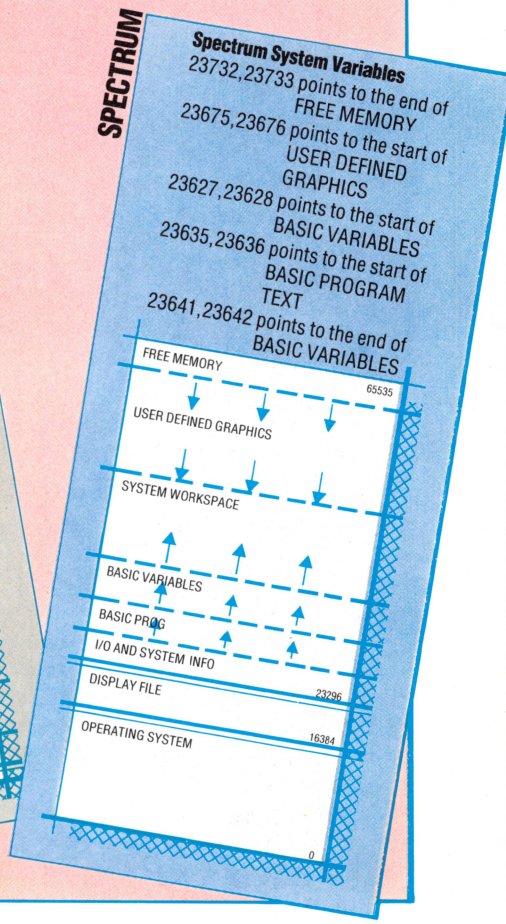
8*256 + 0 = 2048

The fourth and fifth columns in the display are the offset and page bytes for the end of the BASIC Text Area. The address is calculated as:

9*256 + 11 = 2315







BBC Micro

```
7 REM*****************************
8 REM*     BBC      MEMPEEK 1    *
9 REM*****************************
20 MODE 7
30 *TV 255
40 CLS
50 REPEAT
100 INPUT"START ADDRESS ",SA
200 INPUT"NUMBER OF BYTES (0 TO QUIT)",B
N
250 PRINT "*****************************
**********"
300 FOR B%=SA TO (SA+BN-1) STEP 4
350 H$="":@%=6
400 PRINT TAB(0);B%;TAB(8);
450 @%=4
500 FOR C=0 TO 3
550 PK%=?(B%+C):PK$="."
600 PRINT PK%;
650 IF PK%=13 THEN PK$=CHR$(124)
700 IF (PK%>31) AND (PK%<128) THEN PK$=C
HR$(PK%)
750 H$=H$+PK$
800 NEXT C
850 PRINT TAB(32);H$
900 NEXT B%
950 UNTIL BN=0
1000 REM*************************
```

Spectrum

```
7 REM*************************
8 REM*  SPECTRUM  MEMPEEK 1   *
9 REM*************************
30 DIM H$(4)
50 FOR L=0 TO 1 STEP 0
100 INPUT"START ADDRESS ";SA
200 INPUT"NUMBER OF BYTES (0 TO
QUIT)";BN
250 PRINT "*********************
********"
300 FOR B=SA TO (SA+BN-1) STEP 4
350 LET H$="...."
400 PRINT B;TAB 7;
500 FOR C=0 TO 3
550 LET PK=PEEK(B+C)
600 PRINT PK;" ";
650 IF (PK>31) AND (PK<128) THEN
 LET H$(C+1)=CHR$ PK
700 IF PK=13 THEN LET H$(C+1)="■
"
800 NEXT C
850 PRINT TAB 26;H$
900 NEXT B
950 IF BN=0 THEN LET L=2
1000 NEXT L
1050 REM*************************
```

Using Mempeek

When you enter the Mempeek program into your machine, make sure that you SAVE it and check it carefully before you RUN it, because typing errors in this sort of program can lead to unrecoverable crashes.

The program will first ask you for a start address, and then the number of bytes that you wish to examine. Both should be positive whole numbers in the range 0 to 65535. Inputting 0 as the number of bytes will cause the program to end (quit).Suppose you wish the start address to be byte 230. The screen display might look like this:

```
START ADDRESS? 230
NUMBER OF BYTES (0 TO QUIT) ? 8
*********************************
230   193   32   65   49   . A1
234   129   64   93   98   .@]b
START ADDRESS?
```

The leftmost column gives the decimal address of the first byte, the next four columns give the decimal contents of the four bytes from that address on, and the last column gives the character representation of the bytes' contents (where this is possible), and '.' otherwise.

You might like to begin by just 'wandering around' in memory with this program, noting any interesting addresses, and then try to find where in memory the Operating System stores its error messages and BASIC keywords. Your User Manual may help you with this.

Once you've found the pointers that define the boundaries of the various areas of memory, you can try adding some REM lines to the program, and see what effect that has on pointer values. Then add some lines at the start of the program to do some string manipulation, and again, see what effect that has on the pointers and on the contents of the Variable Storage Area.

For example:

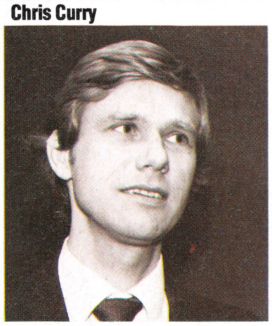
```
3 DIM Z$(254)
4 LET X$=""
5 FOR M=1 TO 255.LET X$=X$+"*":NEXT M
```

Commodore 64

```
7 REM*****************************
8 REM*   COMMODORE MEMPEEK 1    *
9 REM*****************************
30 PRINT CHR$(147)     :REM CLEAR SCREEN
40 PRINT CHR$(142)     :REM UPPER CASE
50 FOR LP=0 TO 1 STEP 0
100 INPUT"START ADDRESS ";SA
200 INPUT"NUMBER OF BYTES (0 TO QUIT)";B
N
250 PRINT "*****************************
**********"
300 FOR B=SA TO (SA+BN-1) STEP 4
350 H$=""
400 PRINT B;TAB(8);
500 FOR C=0 TO 3
550 PK=PEEK(B+C):PK$="."
600 PRINT TAB(8+5*C);PK;
650 IF PK=0 THEN PK$=CHR$(122)
700 IF (PK>31) AND (PK<128) THEN PK$=CHR
$(PK)
750 H$=H$+PK$
800 NEXT C
850 PRINT TAB(32);H$
900 NEXT B
950 IF BN=0 THEN LP=1
1000 NEXT LP
1050 REM*************************
```

FROM LITTLE ACORNS

Chris Curry



Herman Hauser

COURTESY OF ACORN

Acorn Computers is the company that produces two of the finest examples of British home computer technology: the BBC Micro and the Electron. Yet only a few years ago, Acorn was a fledgling enterprise engaged in consultancy work and selling a few special systems from a small office in Cambridge.

Acorn's founder, Chris Curry, was a former employee and close friend of Sir Clive Sinclair. Curry had joined Sinclair Radionics in 1965, when Sinclair had offered him a job as a development engineer for the sum of £11 a week.

At Sinclair Radionics, Curry was put in charge of the project that produced the Executive calculator in 1971. For the next five years, he devoted himself to developing calculators, which are now considered as the precursors of the modern home computer. In 1975 Sinclair Radionics ceased trading, and Curry joined Sinclair in a freelance operation called Science of Cambridge. The new venture aimed to package electronics components as kits.

One idea that sold well was a wristwatch calculator. But Curry was also inspired by the single-board computers that were emerging in the US, and he set about developing a kit of his own. This was called the MK14 (Microprocessor Kit with 14 chips) and featured a National Semiconductor microprocessor, 256 bytes of RAM, a small fixed memory containing the monitor, and the components needed to power an eight-digit LED display.

Curry found that the company was constantly supplying advice and ideas over the telephone to electronics hobbyists and decided to take on Herman Hauser, a PhD student from Cambridge University, to deal with these enquiries. However, Curry's ideas were soon diverging from Sinclair's and he decided it was time to start a company of his own. With Hauser as his new partner, Curry formed a company called the Cambridge Processor Unit — a rather mischievous name if you abbreviate it to CPU! Working from a small office in Bridge Street, Cambridge, the men hired themselves out as electronics and computer consultants.

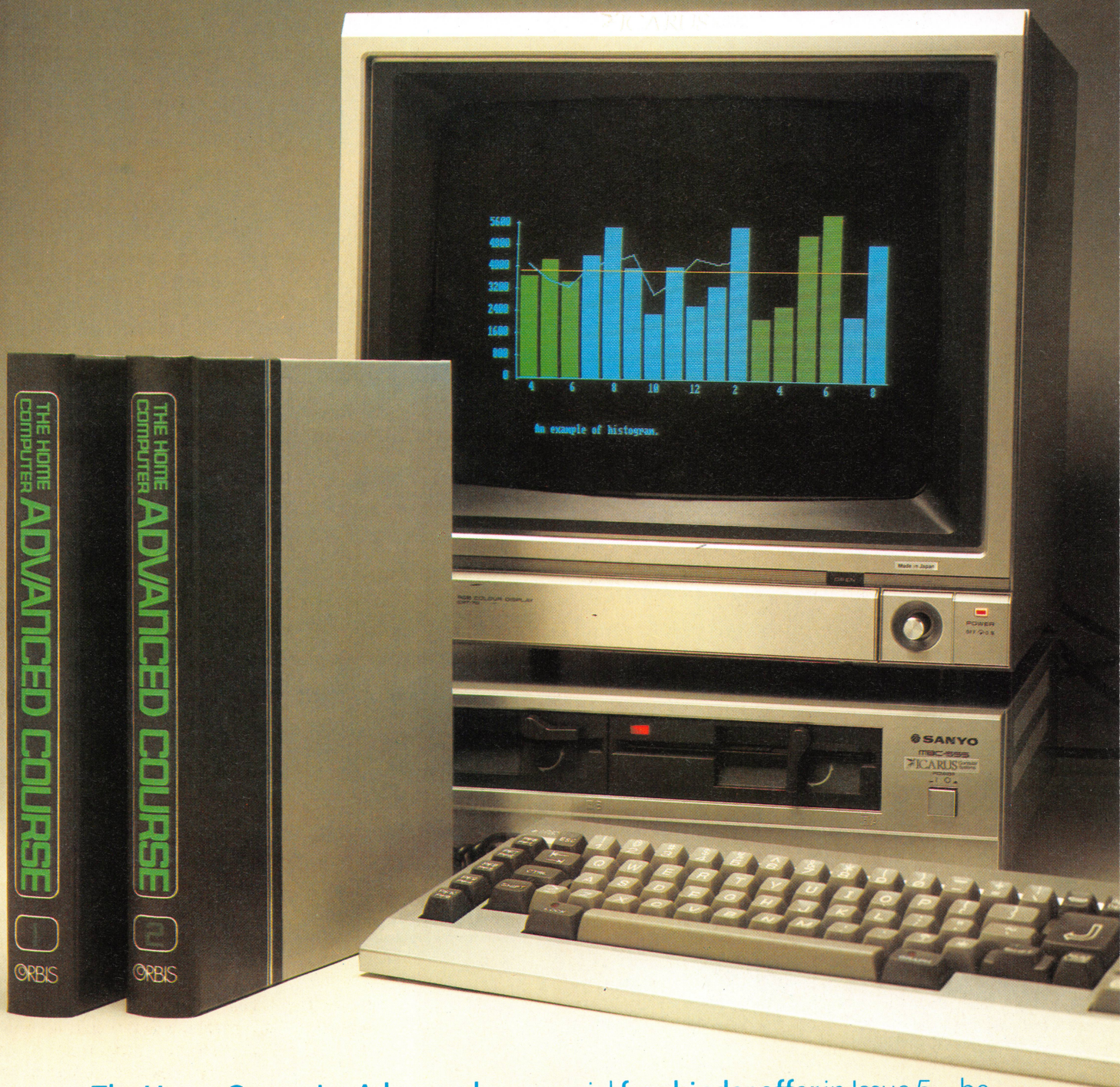
The success of the MK14, and developments in the US, had clearly shown that what the customers wanted was a computer in a box with BASIC on board. Having written a fast version of BASIC for machine control for one of its consultancy projects, CPU decided to add this to a machine and put it on the market. The machine was called the Atom, and CPU adopted the name Acorn for the company that was to market it. The machine was primarily intended to capture the educational market, but most schools thought that the BASIC deviated too much from the Microsoft dialect to be acceptable. The machine did, however, find much favour among hobbyists. Acorn went ahead with a development of the Atom, which was called the Proton and was intended for use in laboratories and colleges.

But in 1981, while the Proton was in pre-production, Curry heard that the BBC was searching for a machine that would complement its computer literacy programme. Curry's response was to demonstrate the capabilities of the 6502 processor — not in the Proton, however, but in a specially designed system.

The BBC specification was for a machine that beginners would find easy to use and yet could be expandable to a very high standard. The machine should also offer good value for money (the Corporation was initially specifying a target price of £200). Against stiff opposition from Sinclair Research, Acorn was given the job, and it created the BBC Micro, which it is now producing at the rate of 12,000 a month.

COURTESY OF ACORN

The volume 1 binder can be yours free!



The Home Computer Advanced Course will take you far beyond the novice stage, widening your knowledge and making you a more sophisticated user.

To help you keep your copies immaculate, we will be making a very special **free binder offer** in Issue 5 – be sure not to miss it!

Overseas readers: this special offer applies to readers in the U.K., Eire and Australia only. Binders may be subject to import duty and/or local tax.

A dream computer, A dream holiday,

and a sight for twenty sore eyes.

WIN – 1st Prize, an I.B.M. P.C. system to the value of £5,000.

WIN – 2nd Prize, a holiday for two in the U.S.A. with a visit to Silicon Valley.

WIN – 3rd Prize, one of ten high resolution colour monitors.

ALL YOU HAVE TO DO

"AH! DRESS THE APE RIGHT,"
EXCLAIMED AMOS, BUGLE SALES MANAGER.

This crazy sentence contains anagrams of four familiar pieces of computer jargon.

Each week until issue 4 we will be featuring the particular parts of the above sentence that make up one of the four words or phrases.

This week (issue 3), the well-known computer word or phrase is hidden in the words "DRESS THE APE"

HOW TO ENTER

To enter the competition, first you must correctly identify each week until issue 4 all four hidden words or phrases. Then use your skill and knowledge of computing *and* your sense of humour to invent a new piece of computer jargon, no more than 5 words long, and give its definition in no more than 15 words.

REMEMBER

Retain your answers to issues 1, 2, 3 and 4. Issue 4 will contain the address and rules for competition entries.

THIS COMPETITION IS OPEN TO READERS IN THE U.K. AND EIRE ONLY